University of Twente

department of
Electrical Engineering

Real Time Control on CAN

Larik-Jan Parchomov
Individual Design Report

Creator: Larik-Jan Parchomov
Supervisors: Dr. Ir. J.F. Broenink
Ing. M.H. Schwirtz
B. Orlic

April 2002

Report 004CE2002
Control Laboratory
Electrical Engineering Department
University of Twente
P.O.Box 217
7500 AE Enschede
The Netherlands

Summary

The goal of this project is to get insight in the use of CAN (Controller Area Network) as a field bus for real-time control. A characterisation has been made of the CAN bus by testing a point-to-point connection between two CAN development boards. Also a third node has been introduced to the network to measure arbitration and delays caused by the bus being ‘busy’.

The results show that the CAN bus is capable of transferring data at a maximum speed of 1 Mbps. With protocol overhead the effective data transfer speed is about 60 to 70 % of the maximum speed, so ‘useful’ data can be sent at a speed of 600 to 700 kbps.

Arbitration takes place by labelling CAN messages with certain priorities. The messages with the highest priority will be sent first. If the bus is busy, the transfer of messages is delayed, even if the message to be sent has a higher priority.

To provide better real-time behaviour, time-triggered CAN should be compared to the results of this project.

Samenvatting

Het doel van dit project is om inzicht te krijgen in de mogelijkheden om CAN (Controller Area Network) te gebruiken voor real-time control. Hiertoe is een karakterisatie van CAN gemaakt door het testen van een point-to-point verbinding tussen twee CAN development boards. Daarnaast is het principe van arbitrage gemeten. Een derde knooppunt is aan het netwerk toegevoegd om de invloed van het ‘bezet’ zijn van de bus op overdrachtssnelheid van data te meten.

The resultaten laten zien dat CAN data kan overzenden met een maximale snelheid van 1 Mbps. Door protocol overhead blijft ongeveer 60 tot 70 % over van deze maximale snelheid voor effectieve (werkelijke) dataverzending.

Het arbitrage principe werkt met het toekennen van prioriteiten aan databerichten. Het bericht met de hoogste prioriteit wordt als eerste verzonden. Als de bus bezet is wordt de datatransfer vertraagd, zelfs al heeft het te verzenden bericht een hogere prioriteit dan het huidige bericht dat verzonden wordt op de bus.

Het verdient de aanbeveling om time-triggered CAN te vergelijken met de resultaten van dit onderzoek, aangezien de theorie uitwijst dat de real-time werking beter moet zijn.

Preface

Almost one year ago I started an individiual design project about real time control on USB2. Along the way many thing changed. The original plan was evaluate and research the control over the USB2. (Universal Serial Bus). Due to delivery problems the focus changed to CAN. The setup of the project remained the same, only the practical part of this project has been done with a CAN bus instead of USB2.

I would like to thank Jan Broenink and Marcel Schwirtz for their patience and support during my project and I would like to thank Thiemo van Engelen for the help on programming microcontrollers.

Larik-Jan Parchomov
April 2002

Table of Contents

List of Abreviations:

USB: Universal Serial Bus
CAN: Controller Area Network
CNI: Communications Network Interface
TTCAN: Time Triggered CAN
CSMA/CA : Carrier Sense Multiple Access Collision Avoidance protocol
PAR protocol Positive Acknowledgement or Retransmission protocol.
UART: Universal Asynchronous Receiver Transmitter
TDMA: Time Division Multiple Access
TTP: Time Triggered Protocol (TTP).
NRZ: Non Return to Zero
RTR: Remote Transmission Request

1. Introduction

This project is about real-time control on field busses. This first chapter gives a description of the assignment, as well as a context for this project. The development path will be focused on and finally a glance forward to this report will be given.

Context

The project is placed within the embedded control systems researched by the control engineering. This research project has been divided in three subprojects (Engineering, 2002), where *real-time control on field-bus interconnected heterogeneous systems* is one of these projects. Some information as can be found on the website (Engineering, 2002)

The purpose of this subproject is to introduce the compositional programming techniques in the context of real–time embedded systems composed of several co–operating processors in networked environments. Application areas are for instance automotive, production machines and Personal Area Networks (PAN's). The basic CSP–framework as being developed in the *software framework* subproject will be used, to simplify the software design process by hiding threads and priority indexing. Moreover, the notion of scheduling is no longer connected to the operating system but has become part of the application instead. This alleviates the distributed software writing problem significantly. Also, reasoning about correctness can be done, because CSP is a formal method.

Furthermore, due to the co-operation with industry in this STW-PROGRESS funded subproject, the feasibility of CSP channels on these systems by means of sophisticated, real-life industrial demonstrators will be shown.

In this individual design project the CAN bus will be explored to get insight whether it is suitable to use it for real time control. In the next section more specific details about the assignment will be discussed.

Assignment

The assignment of this project is real time control on a field bus. This description has two elements which should be focused on: real time and field bus. 'What is real time?' is a question one can ask. What aspects are relevant in relation to real time? Chapter 2 of this report should give some answers to these questions.

The second element to this assignment is field bus. 'What is a field bus?' 'What kind of field busses do exist?' some research questions to be answered. The initial focus in this project was on USB2. This new serial bus has the ability to send and receive data on a maximum speed of 480 Mbps. This high-speed connection is a good starting point to send data fast (real time?) from one point to another. But is it suitable for real time control? To answer these questions the protocol overhead of the connection should be studied. The theoretical studies done on USB2 and CAN can be found in chapters 2 and 3. This theory is based on the theory provided by Hermann Kopetz (Kopetz, 1997)The theory studied should be evaluated in hardware. Therefore a test network should be set up.

Development Path

The first part of this project consisted of a theoretical research in real time concepts and a specific focus on USB2. This research can be found in appendix 1. The next thing to do was to make a test on USB2. After the research on the available hardware, the delivery of the hardware took almost half a year. Before the arrival of the USB2 hardware, the decision had already been made to change to CAN (Controller Area Network). This field bus, mainly used in the car industry, exists longer and has more development

hardware available. But before using this hardware, some theoretical base on CAN had to be founded. This is done in chapters 2 and 3. Chapter 2 describes real time parameters and chapter 3 focuses on CAN specifically. After studying the theory, hardware had to be acquired to do testing. Chapter 4 explains the choice for the hardware as it used during this project.

The final tests consist of a characterisation of CAN, by means of a point-to-point connection. Speed, payload and different configurations give an overview of the abilities of the bus. The second part consists of the bus functionality. Since CAN is a bus, the use of it is in a network. Different nodes use to bus to communicate. This results in moment that the bus is ‘busy’. This has influence on the real time aspects of the communications. By creating a network with three nodes, this influence is tested. The method and results are given in chapter 5.

Theory and practical tests should finally be evaluated. This evaluation should give a picture on how CAN behaves in terms of real time control. This is done in the final chapter.

2 Real-time communication requirements

Introduction

This chapter will focus on real-time networks. The distributed network will be explained in the first section. The second one will summarise the requirements for real-time communication. The theory as provided in this chapter is based on chapter 2 and 7 of Hermann Kopetz' book. (Kopetz, 1997)

2.1 Distributed network

The network that will be focused on is a distributed system. Distributed can be best explained by figure 2.1

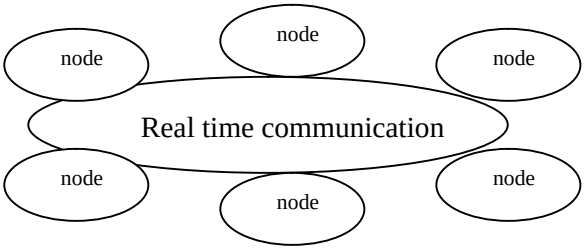


Figure 2.1: Distributed network

Within this network the nodes can be viewed as subsystems. Each subsystem is divided in a host computer, communications network interface (CNI) and communication controller. Figure 2.2 gives a schematical overview of these subsystems.

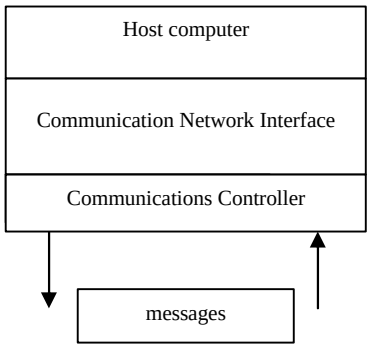


Figure 2.2: the subsystems

The rest of this report will focus on the communication between the various subsystems. The next section will give an overview of all the parameters that have an influence on the real-time communication.

2.2 Real-time parameters

Protocol latency & jitter

Protocol latency is the time interval between start of transmission of a message at the CNI (Communication network interface) of the sending node and the delivery of this message at the CNI of the receiving node. A real-time protocol should have a predictable small maximum of protocol latency and a minimal jitter.

The standard communication topology used in real-time systems is multicast, not point-to-point. Multicast implies that messages are send across the network to all nodes at the same time through the bus. Each node evaluates the message to check whether the message is destined for the node.

Composability

The architecture of the network is said to be composable (pag 146, (Kopetz, 1997)) if a specific property of the network does not change when the node is encapsulated. By encapsulating one of the nodes in the network around the operation of the host, the communication system can become autonomous. Thus, communication through the encapsulated nodes is devoted to the specific operation. The encapsulation isolates any control signals from the CNI.

By integrating the server (communication controller) in the node, the system can also become autonomous (no control signals from CNI), but there are some restrictions. The clients to the node may not overload the server with information, else the real-time condition cannot be guaranteed. One of the solutions to overload of information is to apply flow control to the communication system.

Flexibility

Many real-time communications must support different system configurations that change over time. The restriction in bandwidth causes the system to have constraints of having increase in data traffic.

Error-detection

The communications system must be reliable. Therefore the errors occurring during message transmission should be corrected with minimal protocol latency and jitter. A failure in a node should be noticed and also reported to other nodes. The detection of errors at nodes, for both sender as receiver is called *membership service*. The membership service is fault detection at a certain moment in real-time at which errors are detected at the connected nodes.

End-to-end acknowledgement: The acknowledgement of success of transmission can come from different nodes. Since the information send has an influence on the environment that is monitored by a different sensor/node, the acknowledgement can ‘go-around’. This implies that the information send at node A does not have to be acknowledged by the same node A, but can also be acknowledged by a different node B which responds to the change of the environment caused by the information send to node A.

Physical structure

The physical structure of the communication network depends on economical and technical factors. A multicast communication network requires a bus or a ring network. A point-to point network which has N nodes, requires N-1 communication ports. Therefore the costs are much higher.

Bus versus Ring: In an automotive environment the bus has an advantage: the interface is much simpler and it has the possibility of transferring messages simultaneously to the nodes. In a point-to-point structure the ring has an advantage, since the construction of the ring is much simpler than the bus.(pag 149, (Kopetz, 1997))

2.3 Flow control

Introduction

The control of speed of information between sender and receiver is called the flow control. The receiver determines the speed of the information flow.

Explicit flow control

In an explicit flow control situation, the information send is acknowledged by the receiver. The receiver can use *back-pressure* on the sender to control the flow. This means that the data transfer rate is controlled by the response of the receiver on the send information. The best known protocol in explicit flow control is the PAR protocol (Positive Acknowledgement or Retransmission). This protocol has two parameters: It works with: the time-out interval and the retry-counter. After sending the first time, the retry-counter is set to zero and the sender awaits response. If acknowledgement is given within the time-out interval, the sender continues with the next message. If not, the sender will re-send the message and increases the retry-counter. The sender will keep re-sending until the maximum number or retries has been reached. The *protocol jitter* can be calculated by the maximum number of retries multiplied with the time-out time.

Implicit flow control

With implicit flow control the sender sends and the receiver receives at given points in time. There is no acknowledgement of the transmitted messages. The receiver must detect erroneous transmissions. A way of error checking is to send a number of copies of one message. If at least one of the copies arrives at the receiver, the transmission is successful. The communication is unidirectional and very well suited for multicast transmissions.

Trashing

Trashing is the phenomenon that occurs at a certain point in the throughput of data. As the offered load is increasing, the handled load increases also. At the saturation point the data throughput is maximal. But if at a certain point the throughput decreases rapidly and the offered load is still increasing (and the load is smaller than 100 %), the system is trashing. An example of thrashing can occur in the PAR protocol. As the load increases and the PAR protocol cannot handle the offered load, it causes an additional load as it reaches the time-out interval and starts re-sending messages.

It can be concluded that an explicit flow control is sensible to thrashing. The implicit flow control generates the control signals in time at a constant rate and is therefore not sensible to trashing.

The most important terms for real-time control have been identified in this section. The next section will focus on the architecture of real-time networks.

2.4 Real-time communication architecture

Introduction

This part of the report will focus on the structure of the communication network.

Types of communication networks

Three types of communication networks are distinguished (pag 156, (Kopetz, 1997))

1. The field bus.
2. The real-time network
3. The backbone network.

The *field bus* and *real-time network* must provide guaranteed temporal performance.

The field bus interconnects the nodes in the network to the sensors and actuators in the controlled object. The node often controls the bus and the sensors and actuators are often controlled by an UART (Universal

Asynchronous Receiver Transmitter) interface. The conditions latency and latency jitter are very strict. The field bus should provide clock synchronization. Fault tolerance is not a major issue in the field bus, since the sensors and actuators cause more errors.

The *real-time network* must provide for reliable transmissions with low latency and minimal latency jitter. The network must also provide fault tolerance and must have membership service. The real-time network must have multiple replicated communication channels. This is to prevent the system to crash at one occurring error: *single point of failure*. The real time network must also prevent the *babbling idiot failure* (a node sending messages outside the specified time interval)

The *backbone-network* has the purpose of exchanging non time-critical information with the real-time and production systems. An example of this system is a production report.

2.5 Fundamental conflicts in protocol design

This section gives an overview of the existing conflicts between the various real-time parameters.

External control versus composability

If a group of nodes is connected with each other and composability is applied, the network gets the following constraints:

- The CNI is fully specified in the temporal domain.
- The interconnection of the nodes does not lead to different temporal properties.
- The temporal properties of a isolated host can be tested

If any of these requirements are not met, composability cannot be achieved.

Flexibility versus error detection

Flexibility implies that the behaviour of a node is not set a priori. In a architecture without replication, the error detection is done by the a priori knowledge of the node.

Sporadic data versus periodic data

In architecture with periodic data, the transmission must take place with a minimal latency jitter. The moments of data transmission are known. If at these moments sporadic data is transmitted, the system has to make a choice between the offered data.

Single locus of control versus fault tolerance

If a protocol relies on a single locus of control, it has a single point of failure. In token buses, if the node holding token fails, the communication will fail until the token loss is detected. This problem can also be found in a multi master versus single master control.

Probabilistic access versus replica determinism

If a correct transmission is checked by random properties of the transmission, these will not guarantee that the same transmissions are accepted at different nodes. Vice versa may happen also and the result can be different ‘correct’ transmissions, resulting in functionality failure.

2.6 Media access protocols

Introduction

The media access protocol determines which node should be accessed and how. This protocol represents many properties of distributed real time network. This chapter describes various media access protocols.

A communications is described by two properties:

- Bandwidth: the quantity of bits that can be transferred at a unit of time
- Propagation delay: the time for one bit to travel from one point to the end point of the channel.

The *bit length of a channel* is determined by the number of bits that can travel through the channel within one propagation delay.

The *data-efficiency* is defined by: $data - efficiency < \frac{\#messagebits}{\#messagebits + bitlength}$ (2.1)

2.7. Various media access protocols

This section summarises some well-known media access protocols. The following protocols will be focused on:

- CSMA/CA- CAN
- Profibus
- ARINC 629
- FIP
- TDAM - TTP

CSMA/CA –CAN

The CAN (Controller Area Network) is a typical example of Carrier Sense Multiple Access Collision Avoidance protocol (CSMA/CA). This protocol has the following format:

Arbitratio	Control:6	Data Field (0-64)	CRC (16)	A:2	EOF:7

Figure 2.3 protocol format

Token bus: Profibus

This Token-bus protocol is applied in a token bus system. This is a system, where transfers are controlled by a special control message, a token. The Profibus protocol determines which node has the right to transmit a control code, a token. The loss of a token is a serious error in this system.

Minislotting: ARINC 629

With minislotting, the time is partitioned in minislots of time, which are longer than the propagation delay of the channel. The ARINC 629 protocol works with ‘stages’. In the first stage the set of processes that wants to transmit are collected. In the next phase, called an epoch, all processes may transmit their messages.

Central master – FIP

The central master protocol works with a central control of the busses. In case a central node fails, another node takes over. The FIP protocol is a master control protocol. The FIP contains a static list with names and periods of variables. As the master broadcasts these variables, the node answers and the variable is set.

TDMA –TTP

The Time Division Multiple Access principle is based on the global time, which is divided in a number of slots. A unique slot is assigned to a node. In this way, various messages can be transmitted simultaneously. The Time Triggered Protocol (TTP) uses this principle.

Event triggered versus time triggered

An event-triggered implementation sends an event message as soon as an alarm has been recognized. A time-triggered implementation sends a periodic state message every period. As the number of alarms

increases the generated load increases at the event-triggered implementation. The time-triggered implementation remains at the same load.

Now the basic parameters for real-time control have been introduced, the focus will change to CAN. In the next chapter an introduction is given to CAN and its application area. Together with this chapter it should form a theoretical base and the knowledge should be sufficient to get started with CAN hardware.

Chapter 3 Introduction to CAN

Introduction

CAN (Controller Area Network) is a serial communication bus for real time applications. CAN was originally developed for in-car use. Nowadays most modern cars use in-car management based on CAN. The area of application is also shifting to other industries and CAN is applies in industrial control systems and embedded networks .(CIA, 1999a)

This document will give an overview of CAN. In the first part some examples of applications will be given. Next the CAN Datalinklayer and physical layer will be focused on. The last part of this document consists of the specification of the CAN 2.0b and also gives an overview of Time-Triggered CAN (TTCAN). The information found in this chapter is a summary of the information that can be found in various document on the CAN in Automation site (Automation, 2002)

3.1. Applications

Daimler-Benz was the first car producer which implemented CAN in the engine management. Today cars and trucks made by Daimler Benz use CAN for engine management at a speed of 500 kbit/s. Besides the application of engine management, CAN is also used for electronic control units. These control units use CAN at a lower speed, about 125 kbit/s. The third application in cars for which CAN is used, is entertainment. In the following figures an illustration is given of the use of CAN.

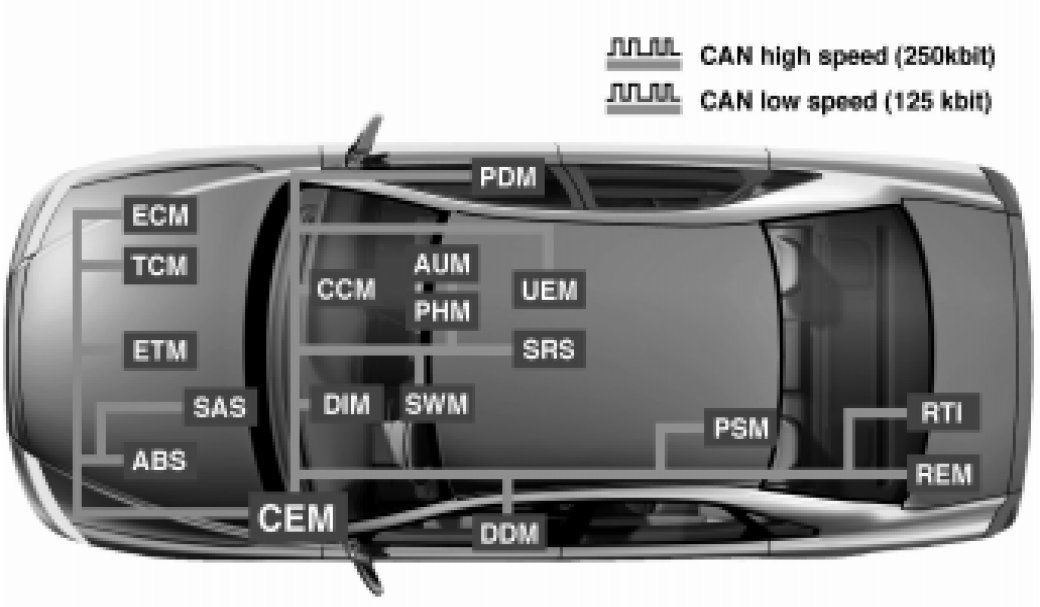


Figure 3.1: Can used for multimedia (CIA, 1999a)

Besides use in cars CAN is also applied in industrial control systems. Examples of machines which use CAN are printing and posting machines, bending machines, wood processing machines and semiconductor manufacturing machines. Also robots are controlled by a CAN network. In this network the servo controllers, the control units and several I/O devices are connected to each other.

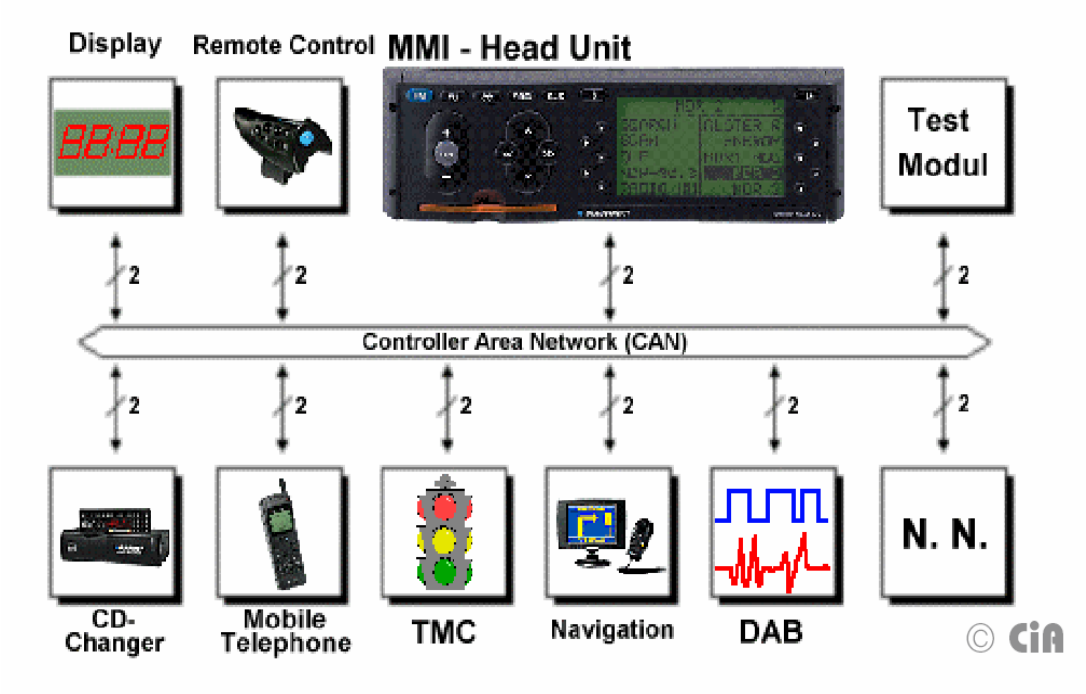


Figure 3.2: CAN used in multimedia (CIA, 1999a)

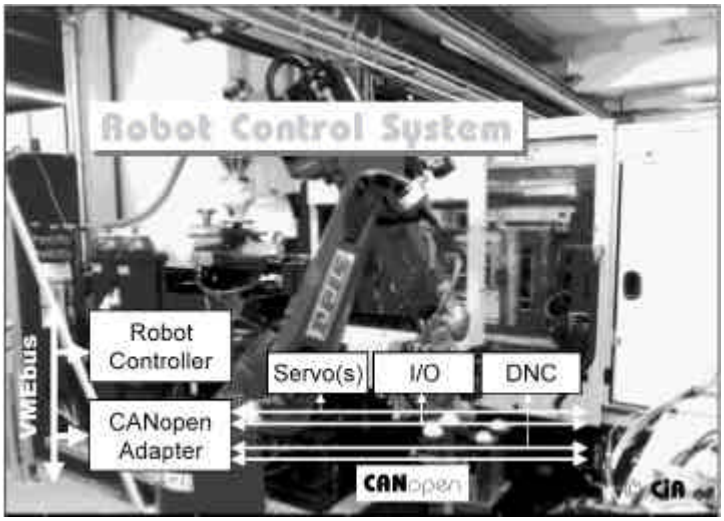


Figure 3.3: Robot controlled via CAN (CIA, 1999a)

The illustration shows a robot which is controlled by a 2 line network. At each network 5 servo drives are controlled by a synchronization time of 10 ms. At a baud rate of 500 kbit/s this resulted in a bus load of 60%.

3.2. Physical Layer

Introduction

This part will explain more about the physical layer of the CAN network. The physical layer consists of the data transmissions between the nodes. This section will discuss the requirements of the physical layer and some of the existing concepts will be mentioned.

Physical Signalling (PLS): - Bit encoding, timing and synchronization.
Physical Medium Attachments (PMA): - Transceiver characteristics
Medium Dependent Interface (MDI) - cable connector

Fig 3.4: The CAN physical layer

The CAN physical layer can be divided in three sub-layers. This is schematically shown in figure 3.4 The PLS layer is implemented in the CAN controller chips. The PMA layer describes the transceiver characteristics. The MDI layer specifies the cable and connector characteristics.

The PMA and MDI layers are subject of different international, national and industry standards as well as proprietary specifications. Most common is the ISO 11898 standard specifying a high-speed transceiver for CAN-based networks. In the next part each of the layers will be explained.

Physical Signalling: Bit encoding

CAN uses Non Return to Zero (NRZ) bit encoding. The bit levels are either *dominant* or *recessive*. Manchester coding is also used for encoding. Manchester encoding requires a falling or rising edge in each bit. The encoding can be summarised by figure 3.5. This figure shows that after one bit with recessive bit-level (+5 volt), the bit-level does not return to the dominant level (0 volt).

Figure 3.5: NRZ-encoding [CIA, 1999 #2]

One of the characteristics of Non-Return to Zero encoding is that in a consecutive row of bits with the same polarization, the edges of the bits cannot be detected. For this reason bit-stuffing is used. This means that after a row of five bits with the same polarization an extra bit is added with the opposite polarization. This ensures synchronization between the network nodes. Bit stuffing is illustrated in figure 3.6.

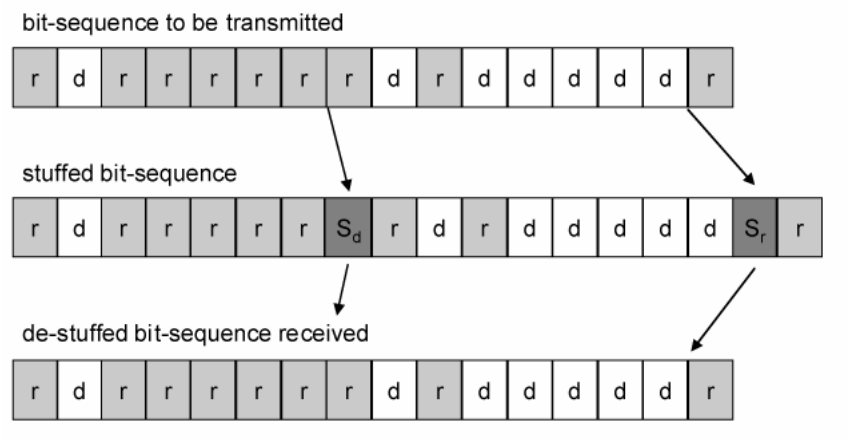


Figure 3.6: Bit stuffing (CIA, 1999c)

Physical Signalling: Signal Propagation

When transmitting data over the CAN bus, propagation delays occur. These delays can be specified by the blocks given in the next diagram.

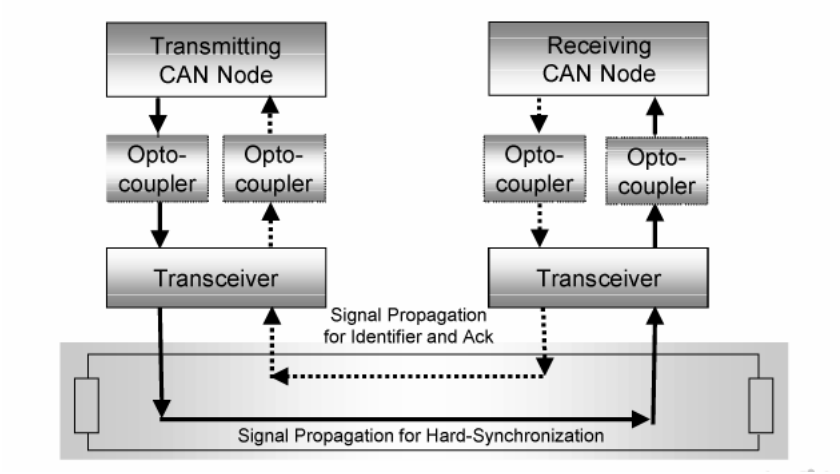


Figure 3.7: Propagation delay overview (CIA, 1999c)

The sum of the propagation delay times consist of controller delay, optional galvanic isolation delay, transceiver and bus line delay:

CAN controller:	50 ns to 62 ns
Optocoupler:	40 ns to 140 ns
Transceiver	120 ns to 250 ns
Cable	about 5 ns/m

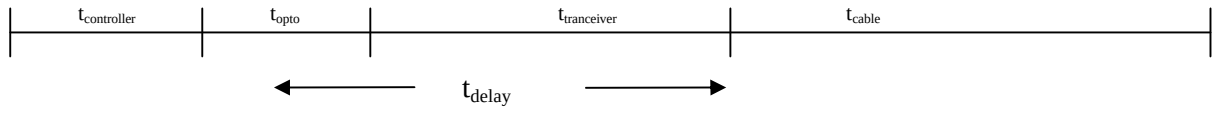


Fig 3.8 Timing diagram of propagation delay

These delays have to be multiplied by two, because after hard synchronization the most the transmitter has to wait for a acknowledgement bit. Using ISO 11898 compliant transceiver and high-speed optocoupler a maximum bus length of 9 meters at 1 Mbit/s can be reached.

As can be seen in the formula for the propagation time the busspeed is dependent of the length of the bus. Table 1 gives an overview of the line-speed dependency (CIA, 1999c)

	Bus length	Nominal bit-time
1 Mbit/s	30 m	1 μs
800 kbits/s	50 m	1,25 μs
500 kbit/s	100 m	2 μs
250 kbit/s	250 m	4 μs
125 kbit/s	500 m	8 μs

Table 3.1: Dependency between bus speed and bus length

Physical Medium Attachments: Nominal Bus levels

The conditions for detecting the right bit can be described by the nominal bus levels. The buslevels are given in figure 3.9.

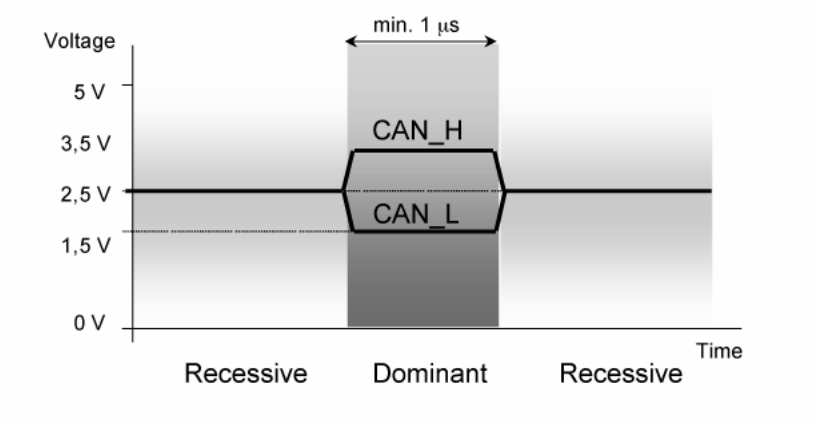


Figure 3.9: Nominal buslevels (CIA, 1999c)

The recessive and dominant levels for the bits on the CAN bus can be described by the difference between Can_L (lowest voltage on CAN) and Can_H (Highest voltage on CAN). If the difference is smaller than 0.5 volts, a recessive bit is detected. If the difference is bigger than 0.9 volts, a dominant bit is detected.

The two buslines CAN_L and CAN_H are used to make the CAN bus insensitive to electromagnetical interference. Since both lines are affected in the same way by the interference, the difference between the 2 signal levels stays the same and thus usable.

3.3 CAN Datalinklayer

The CAN Datalinklayer is standardised in ISO 11898. This part of the document will shed some light on some of the relevant aspects of this ISO standard. This should result in a broader knowledge on how the CAN network operates.

3.3.1 Communications.

The CAN network is a multicast network. This means that every node connected to the network is 'listening' to all the broadcast messages send over the network. Since every message broadcasted is not useful or ment for a certain node, message filtering should be applied. The setup of the CAN network can be found in figure 3.10. (CIA, 1999b)

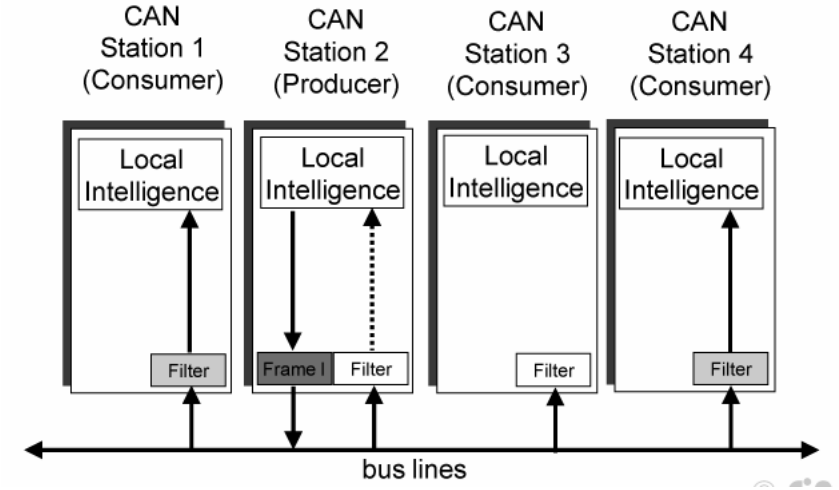


Figure 3.10: CAN broadcast (CIA, 1999b)

Besides sending message to nodes over the network, it is possible for a node to request a message. This can be compared to asking a question over the network. This Remote Transmission Request (RTR) can be followed by a data transmission done by another node: the answer to the question.

Since the CAN network is a multicast network, it is possible for different nodes to access the network at the same time. In this case, so-called Arbitration takes place. The method for arbitration is called Carrier Sense Multiple Access with Collision Detection and Arbitration on Message Priority (CSMA/CD + AMP). If two or more bus nodes start their transmission at the very same time after having found the bus to be idle, collision of the messages is avoided by the implemented CMSA/CA + AMP bus access method. Each node sends the bits of its message identifier and monitors the bus level. As long as the bits from all transmitters are identical nothing happens.

If a bit at node 1 is dominant and at the same time node 2 sends a recessive bit, the recessive bit will be overwritten. This action is noticed by node 2 and instead of sending a message; the node will listen to the network. An example:

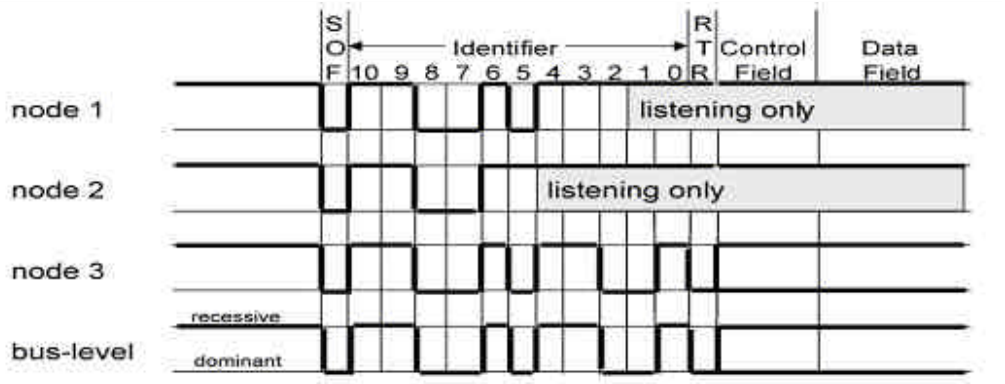


Figure 3.11: Arbitration (CIA, 1999b)

At bit 5 nodes 1 and 3 send a dominant identifier bit. Node 2 sends a recessive identifier bit but reads back a dominant one. Node 2 looses bus arbitration and switches to listening only mode that is transmitting recessive bits. At bit 2 node 1 looses arbitration against node 3. This means that the message identifier of node 3 has a lower binary value and therefore a higher priority than the messages of nodes 1 and 2. In this way the bus node with the highest priority message wins arbitration without loosing time by

having to repeat the message. Nodes 1 and 2 will send their messages after node 2 has finished his transmission.

3.3.2 Data format

There are two data formats available in CAN: version 2.0a and version 2.0b.
The schematic overview of both versions can be found in figures 3.12a and 3.12b.

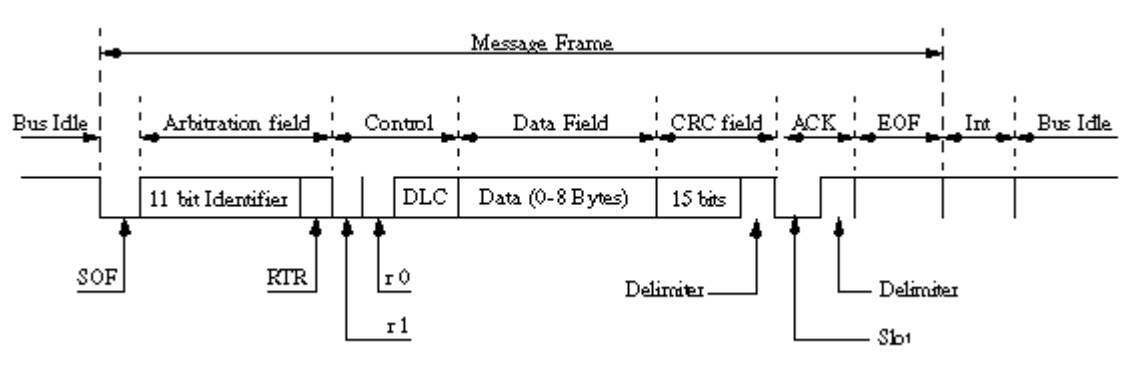


Figure 3.12a: Version 2.0a data frame (Schofield, 2001)

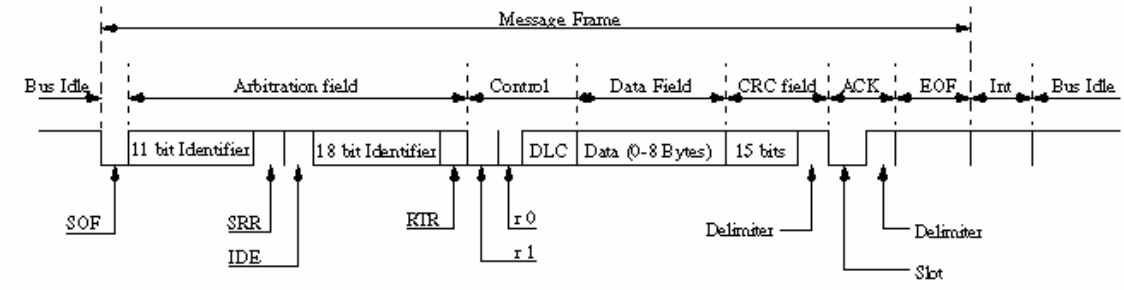


Figure 3.12b: Version 2.0b data frame (Schofield, 2001)

Since the difference between both versions is an extra 18 bit identifier for version 2.0b (to provide more compatibility) both data frames can be explained simultaneously.

The data frame starts with a Start of Frame field. The dominant bit level (logic 0) at this point marks the start of the data frame. The SOF is followed by the arbitration field and the Remote Transmission Request (RTR). The arbitration filed contains the identification bits. The RTR field contains either a recessive bit (logic 1) or a dominant bit (logic 0). A dominant bit level points out that the data frame is a remote transmission request and ends after this field. In case of a recessive bit level the data frame contains data for a specific node. In that case the data frame is continued by the DLC: data length code and the to be transmitted data (0 – 8 bytes). After the data there is a 15 bit cyclic redundancy bit, followed by the delimiter bit. Finally the data frame has a 2 bits ACKnowledgement field and 7 bits end of frame (EOF) field. The first bit of the two Acknowledgement bits is recessive, but in case of successful reception by other nodes the second bit is overwritten to be dominant.

After the transmission of a data frame an INT period of several bit-times follows. The purpose of this time is to let the CAN controllers prepare for the next transmission.

In addition to the version 2.0a data frame the version 2.0b data frame has an extra 18 bits identifier. After the first 11 bits (equal to the version 2.0a format) a SRR (Substitute Remote Request) and an IDE (Identifier Extension) field follows. The SRR field is always recessive in format to ensure in case of arbitration that the version 2.0a data format will have priority to the version 2.0b data frame. The IDE field is to point out if the transmission is done by a version 2.0b or a version 2.0a controller.

The most features and characteristics have been discussed at this point. We will now focus on a new type of CAN, the so-called Time-Triggered CAN (TTCAN).

3.4. Time-Triggered Can

In the CAN network arbitration is used to transfer all messages according to their priority. In the near future CAN will be used in mission critical networks, for example x-by-wire systems in cars. For this area of application additionally deterministic behaviour in communication is required. The point in time when a message is transferred must be predicted. (Führer and Müller, 2000)

The principle of time-triggered CAN is the use of a periodic transmission of a reference message, which offers a system wide global network time with high precision. Based on the network time, messages are assigned to different time windows within the basic cycle.

3.4.1 Focus on the operation.

Time triggered operation means that the activity of a system is determined by the progression of the time. By the use of transfer schedule the sending and receiving of messages can be determined. By the use of this time table the results can be packed in a deterministic and predictable matrix. An example of this kind of timetable can be found in fig 3.14. The necessary information of this matrix can be mapped in each node in the network. (Führer and Müller, 2000)

The CAN protocol uses bitwise arbitration to control the message transfer over the bus. The arbitration assures that a message with high priority will be send earlier than a message without lower priority. The latency of sending the priority message consists of the time that has to be waited to access the bus in case another node is already transmitting over the bus. The latency jitter increases if the priority of a message is lower. The goal of the time-triggered operation is to get rid of this latency jitter. Therefore the protocol specification ISO 11898 has been extended to ISO 11898-4 in 2 levels:

- Level 1 guarantees the time triggered operation of CAN based on the reference message.
- Level 2 is in charge of the globally synchronised time base that is established between the nodes. Drift in time is cancelled out.

The reference message

The periodic communication is controlled by the reference message. At level 1 extension of CAN the reference message contains of one byte of control information. The rest of the message is used for CAN data. At level 2 extension the reference message also holds the global time information of the TTCAN transfer, with the size of 4 bytes. The setup of this reference message can be found in figure 3.13.

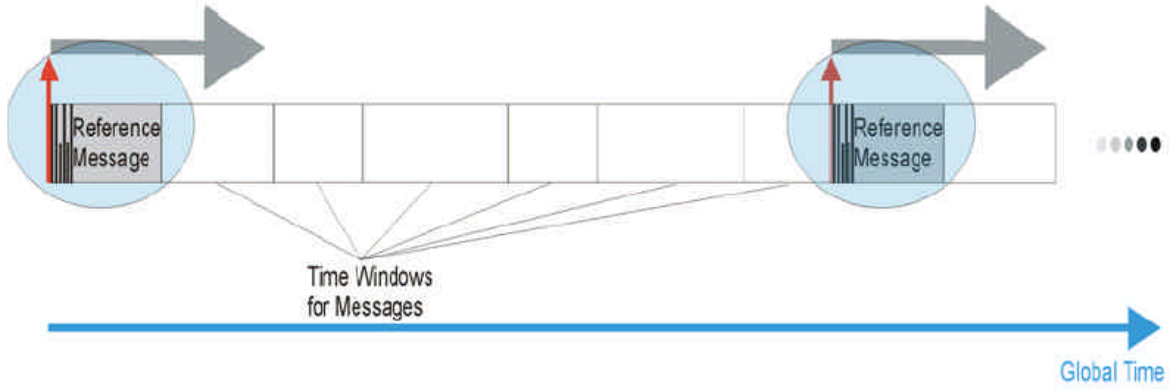


Figure 3.13: The reference message starts the basic cycle (Führer and Müller, 2000)

The basic cycle

The period between two consecutive reference messages is called the basic cycle. The basic cycle consists of the reference message and some time windows of different size in which messages can be send. The time windows can be used for periodic state messages, spontaneous state and event messages. The messages sent, all use the CAN data format. A time window for spontaneous messages is called an arbitrating time window. At this moment the arbitration decides which node can send a message (the same way as in non time-triggered CAN).

The nodes in TTCAN

In TTCAN the controller of the node does not have to know all the messages send over the network. The only necessary information for the node is when to send what message (for example: message A can be send at timestamp 1,3,4, see figure 3.14)

The system matrix

The system matrix usually consists of few basic cycles. Together they form the system matrix. The best way to further explain this is by the picture shown in figure 3.14.

Sending and receiving messages

The sending and receiving of messages at TTCAN is driven by the progression in time. The cycle time of TTCAN is restarted at each reception of the reference message. The so called time marks form the link between the cycle time and the system matrix. These time marks point out the beginning of a time window. Besides the starting point of a time window, the time-mark also consists of a base mark and the repeat count. The base mark indicates the number of the first basic cycle in the matrix cycle in which a message should be sent or received. The repeat count tells the number of basic cycles between two successive transmissions of the same message (see also figure 3.12).

Generation of network time

The cycle time in TTCAN guarantees the time triggered operation of the network. The timing information is given in network time units (NTU). The cycle time is thus given in NTU and is based on the nominal bit time at level 1 TTCAN. At level 2 TTCAN the NTU is based on the so-called physical second (2^{-n}). The network time generation can be schematically pointed out by figure 3.15.

Fig. 3.14: Example of a time triggered Can system matrix. (Führer and Müller, 2000)

Global time generation at level 2 TTCAN

In the TTCAN level 2 networks there is a time master. The time master is the node that sends the reference message. Every time this node sends out a timestamp, this time stamp is correct by definition. Every node synchronises with this time stamp (which is part of the reference message) by taking a snapshot of the time stamp and synchronise in this way. After taking the snapshot each node can calculate its offset by calculating the difference between the global time stamp given by the time master and the local time at the node. At the next basic cycle a node can compute the global time by defining the global

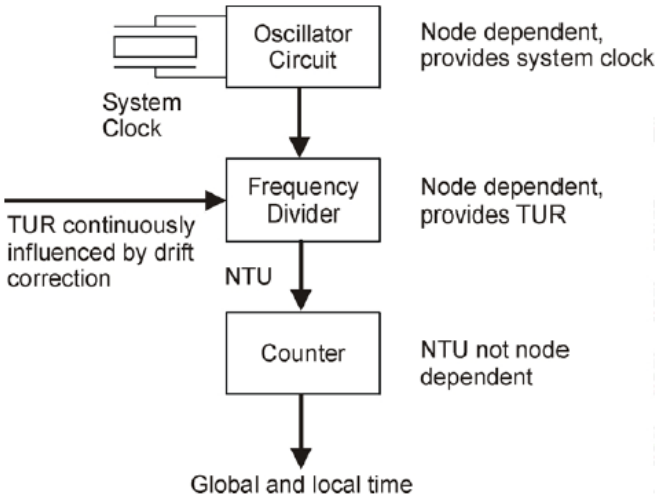


Fig. 3.13: the generation of network time (Führer and Müller, 2000)

time to be equal to the local time plus the local offset. This system works as long as each clock at each node has exactly the same speed. Unfortunately this is not the case so a drift correcting mechanism has to be introduced. This mechanism consists of the continuous update of the time unit ratio (TUR). TUR

corrects the local clock to the network time unit (see also figure 13). TUR is defined by the specification of the node oscillator. During operation an extra adaptation takes place by measuring the (local) frame synchronisation pulse and the global time (i.e. the time between to timestamps given by the time master).

Fault Tolerance of the time master

As we have seen in the previous part, the time master is the heart of TTCAN. Therefore fault-tolerance must be established. This is done by defining another potential time master. The fault tolerance principle works at follows: After reset, the potential time master checks if there is bus traffic and if a reference message has already been send. If not, the potential time master sends a reference message with its identifier and (at level 2 TTCAN) its local time to be the global network time. The other nodes assume this node to be the time master. If a reference message is send over the network with a higher priority (thus from the real time master), the potential time master stops sending the reference message and synchronises to the real time master. In the opposite situation, when a reference message with lower priority is send over the network, the potential time master first synchronises to the real time master and at the next basic cycle it tries to become the time master by sending its reference message. But the arbitration will cause the potential time master to lose the arbitration. By applying this principle in the network, the node with the highest priority will become time master, without violating the structure of the basic cycles.

If a reference message is missing, all time masters will respond to this, by sending a reference message after a time-out. The above described principle will become active again and the functionality has been re-established.

4. Hardware

Introduction

One of the assignment requirements is to make do a test on CAN. With this test the CAN bus should be characterised and the possibilities of real time control should be explored. Therefore hardware is needed of course. This chapter will give an overview of the selection path taken to acquire the most fitted hardware.

4.1 Requirements

To select the right hardware, first the test method has to be defined. To characterise a point-to-point connection at least 2 nodes are required. For simulating a network with different nodes and each node having its own task, a third node is also required. The total network of three nodes is schematically shown in figure 4.1. In this figure the acquiring of samples is taken as the task of the network.

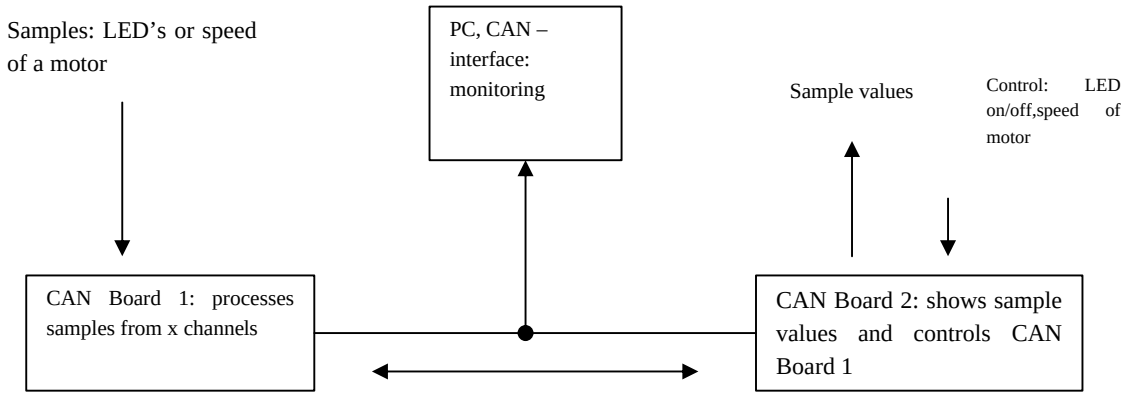


Figure 4.1 Block diagram of test method

Besides the hardware, the software has an important role too. To program an application for the CAN network, one needs a development tool for the microcontroller used in the hardware. It is also required to make the network function on its own, thus without the control of a pc, since the pc is a bad influence when it comes to timing and time-critical tests.

From the hardware and software demands, we can summarise the requirements as done below. The requirements will be used when selecting the development tools.

4.2 The products

With the requirements defined, the search for products starts. Using the internet search sites and the CIA site [Automation, #9] various products are found. From these products a selection is made. The next sections will give an overview of the products and the test-configurations that can be made with these products.

4.2.1 The CAN boards

1) IME-ACTIA Eva Board: IME 2308 205

This is an euroboard with an Infineon C167CR @ 20 Mhz microcontroller. This microcontroller can be programmed via an RS232 interface. The board has the following memory:

- 2 x 256 kb EPROM (16 bit)
- 2 x 128 kb SRAM (16 bit)
- 2 x 128 kb Flash (16 bit)

The board is delivered with on-board control software, manual, circuit diagram, levelX drivers and PCCancontrol software. The block diagram can be found in the figure 4.2.

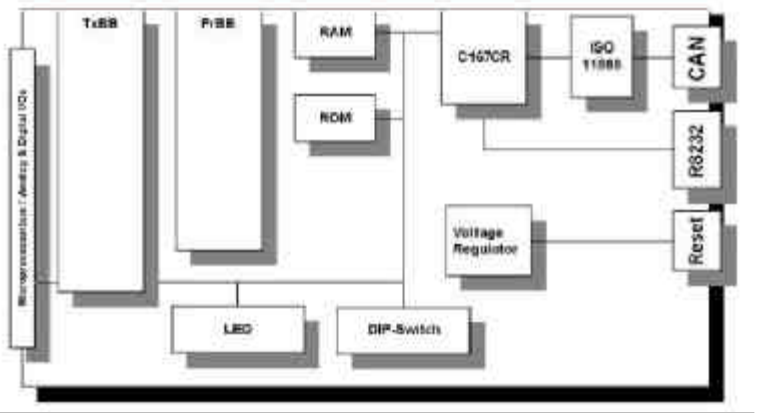


Figure 4.2. Block diagram of Ime Actia Evaboard

2) Phytec T89C51CC01 Rapid Development Kit

This is a single board computer with an Atmel 8-bit 80C51 compatible T89C51CC01 microcontroller @ 20 Mhz. The board can be programmed by an RS232 and or RS485 interface. The board has 128 kb external Flash, 32 kb external SRAM and 4 kb external serial I2C-EEPROM

3) Solution made by Theo Lammerink (LMK board)

The third and final available solution is a board made by Theo Lammerink. The best way to describe this board is by the following block diagram.

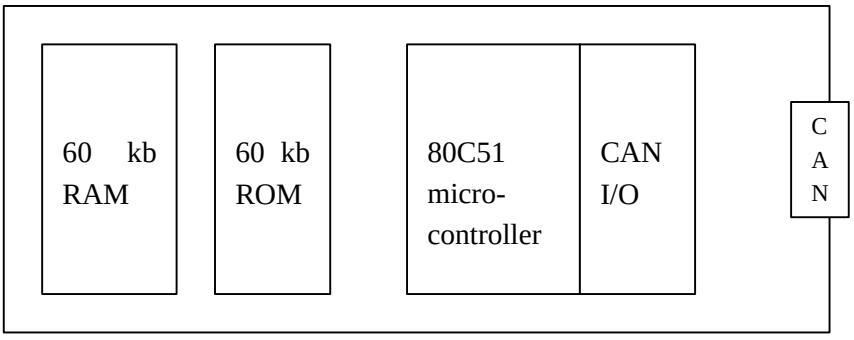


Figure 4.3: Block diagram of the LMK board

This board is based on an 80C51 microcontroller. The 40 I/O ports of this controller have been used for the CAN interface (Philips SJA 1000). The available memory for programming is 60 kb Ram and 60 kb ROM.

4.2.2 PC-interface

Ime Actia has a Can pc interface: Ime Actia PC slimline IME 1007 401. This PC interface has the ability to watch CAN traffic on the bus in combination with the software PC CANCONTROL. Besides this function it can also generate traffic. For these reasons this board is really interesting to use for this project.

The product is an ISA card with a combination of an 80C51 microcontroller with Philips SJA 1000 protocol interface. The card has no interface to program the microcontroller, but the microcontroller is pre-programmed to the ISO 11898 standard for CAN 2.0B. The software provided with this card is PCCANCONTROL. This program enables tracing and selecting of activities in the network. The block diagram of this card can be found in the following figure 4.4.

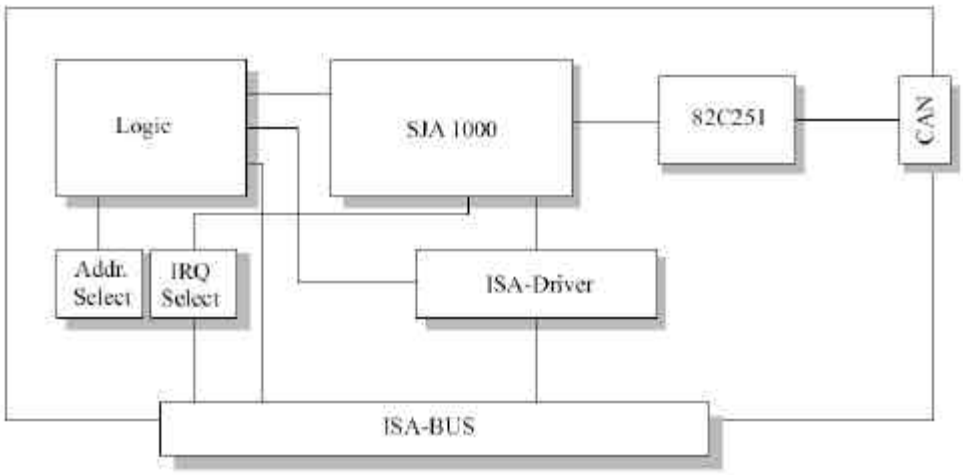


Figure 4.4.: Pc SlimLine blockdiagram.

4.3 Software

4.3.1 PCCanControl

This program can be used on all IME ACTIA boards. The program supports CAN protocols 2.0a and 2.0b. More of this program can be found at the webpage. (Actia, 2002)

4.3.2 KEIL compiler software

KEIL software provides a programming environment for both the 80C51 microcontroller and the Infineon C167 microcontroller. The environment consists of a C compiler and microcontroller specific assembler and debugging tools. The development environment is a Windows 9x/NT based program. More information can be found at the webpage. (Keil, 2002)

4.3.3 Tasking compiler software

Tasking compiler software can be compared to KEIL. The program exists for both microcontrollers. The software consists of a C compiler, assembler and debugging tools. Tasking has a webpage, where more information can be found (Tasking, 2002)

4.4 Configurations

Out of the mentioned products three options for configurations are possible. These are summarised below.

Configuration 1:
PC-interface: IME-ACTIA PCSlimLine
CAN Boards: 2 IME-ACTIA Evaboards
Software: KEIL C166 evaluation software, PCCanControl

Configuration 2:
PC-interface: IME-ACTIA PCSlimLine
CAN Boards: 2 times solution Theo Lammerink
Software: IDE, provided by Theo Lammerink, PCCanControl

Configuration 3:
PC-interface: IME-ACTIA PCSlimLine
CAN Boards: 2 Phytec T89C51CC01 rapid development kits
Software: KEIL C51 evaluation software, PCCanControl

4.5 Costs

At this point the requirements are made and for the project other selection criteria must be used to choose the final configuration. One of these criteria are the costs. In table 4.5.1 an overview is given prices of the products.

Product	Price	Remarks
IME Actia Evaboard	Hfl 1050,- excl.	Price per board
Phytec Rapid Devkit	Hfl 695,- excl.	Price per board
Solution of Theo Lammerink	Hfl 2000,- incl	2 boards
Pc Slimline	Hfl 400,- excl.	Price per board
Keil Software 80C51	Hfl 4800,- excl.	Evaluation version free, limited at 2 kb source code
Keil Software C166	Hfl 6800,- excl.	Evaluation version free, limited at 4 kb source code
Tasking C51 Toolset	Hfl 2800,-	Evaluation version free, limited at 2 kb source code
Tasking C16x Toolset	Hfl 7000,- excl.	Evaluation version free, limited at 4 kb source code

Remark: all prices are dated november 2001

Table 4.5.1: Overview of the prices of the products

4.6 Evaluation

The final selection of a configuration mentioned in section 4.4. is done by various aspects:

- price
- time of deliverance
- documentation
- available software
- type of processor
- memory

Of these aspects we can make an evaluation matrix, found in table 4.1

.

Configuration Nr	Description	Cost (btw incl., shipping excl)	Time of delivery	Software	Microprocessor	Memory	Documentation
1	PC Slimline + 2 IME Actia Evaboards	Hfl. 2903,-	1 week	KEIL C166 eval, PCCAN CONTROL	Infineon C167CR @ 20 Mhz	2 x 256 kb EPROM (16 bit), 2 x 128 kb SRAM (16 bit), 2 x 128 kb Flash (16 bit)	Manuals, Datasheets, diagram layout
Evaluation		+/-	++	++	++	++	+
2	PC Slimline + 2 boards provided by Theo Lammerink	Hfl. 2476,-	1 week	IDE , PCCAN CONTROL	80C51, SJA 1000	60 kb ROM, 60 kb RAM	No documentation, only Theo Lammerink's knowledge
Evaluation		+	++	++	+	+	+
3	PC SlimLine + 2 Phytec evaboards	Hfl. 2130,-	2 weeks	KEIL C51 eval, PCCAN CONTROL	T89C51CC01 @ 20 Mhz	128 kb external Flash, 32 kb external SRAM, 4 kb external serial I2C-EEPROM	Manuals, Datasheets, diagram layout
Evaluation		++	+	++	+	+	+

Table 4.1: Evaluation matrix

4.7 Conclusion

Configuration 1 is the preferred configuration, because of the quantity of memory, the stronger microcontroller of Infineon and the products are all from the same supplier. With this configuration the CAN bus characterisation can take place.

5. Measurements & Results

Introduction

This chapter is divided in three parts: looptime measurement, bus arbitration and busload influence. Each of these sections refers to the measurements done. Every measurement description is also divided in three parts: measurement setup, measurement results and discussion (of the results).

Remark: In order to use the complete Tasking solution with IME ACTIA CAN boards a manual has been added to this report. In appendix 4 this manual can be found.

5.1 Point-to-point connection

Introduction

This first section describes the measurements done on the point-to point connection.

5.1.1 Point-to-point connection: measurement setup

The characterisation done on the CAN network is done by the use of a point-to-point network. Goal is to get insight in factors that have influence on speed the CAN bus. These factors can be summarised by the following questions:

- How does the payload of a CAN message influence the transfer speed?
- What is the ratio between bus speed and payload?
- What is the ratio between the bus speed and the transfer time of a CAN message?

By answering the questions mentioned, it should be possible to give a characterisation of a point-to-point connection over the CAN bus. The measurement of the various factors is done by the use of a sender and a receiver, both represented by a CAN development board. The boards are connected to each other by the CAN bus. The sender is also connected to the pc, with which the results of the measurement are read and saved. The total testing environment can be showed by the block diagram in figure 5.1.

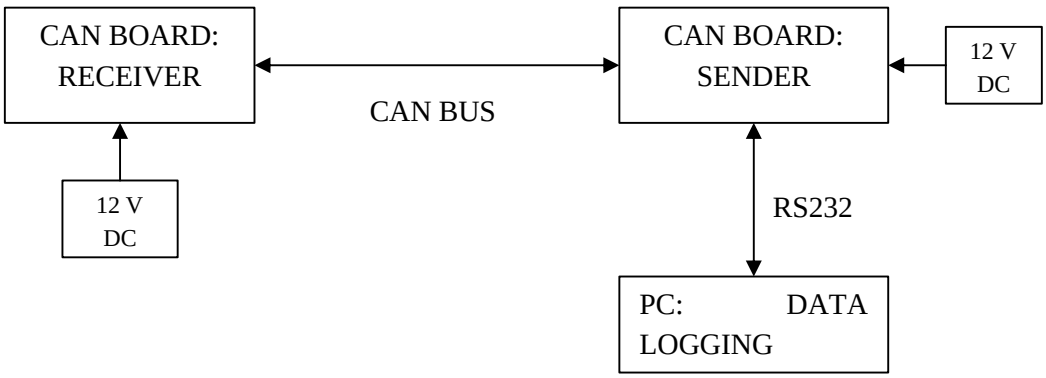


Fig. 5.1: Block diagram looptime measurement

The basic measurement consists of the calculating the so-called ‘looptime’ of a CAN message. This looptime starts at the moment the sender sends a CAN message. After reception the receiver returns this message to the sender. At the moment the sender receives the message, the looptime measurement is complete. By varying the payload and the bus speed, it is possible to give a characterisation of the bus transfer by means of this looptime.

The looptime consists not only of bus transfer time (from sender to receiver and vice-versa), but also of some CPU time of the receiver and sender. The CPU time of the receiver is also measured. At the moment the receiver receives the CAN message a timer is started. This timer is stopped at the moment of

sending the message back to the sender. The value of this timer is put in the payload and is transferred through CAN to the sender. The sender saves this value and shows this value on the RS232 port after the measurement.

The easiest way to explain the way of measurement is by way of finite state machine. The finite state machine for the sender is given in figure 5.3. on the next page. After the finite state machine of the sender, the one of the receiver can be found in figure 5.4.

The source code has been programmed in the way the finite state machines describe in figures 5.3 and 5.4. First an initialisation is programmed. In this part the speed of the bus, the payload and priority of the message is programmed. After the initialisation the functions are programmed for sending and receiving the test-messages. Since the evaluation version of the Tasking compiler was used, it was not possible to make one program to test the looptime for all the speeds and all the different payloads. So for each payload (1 byte till 8 bytes) a different project had to be made. The code used for different payloads than 1 byte is almost the same, only in the initialisation part of CAN messages some code is different.

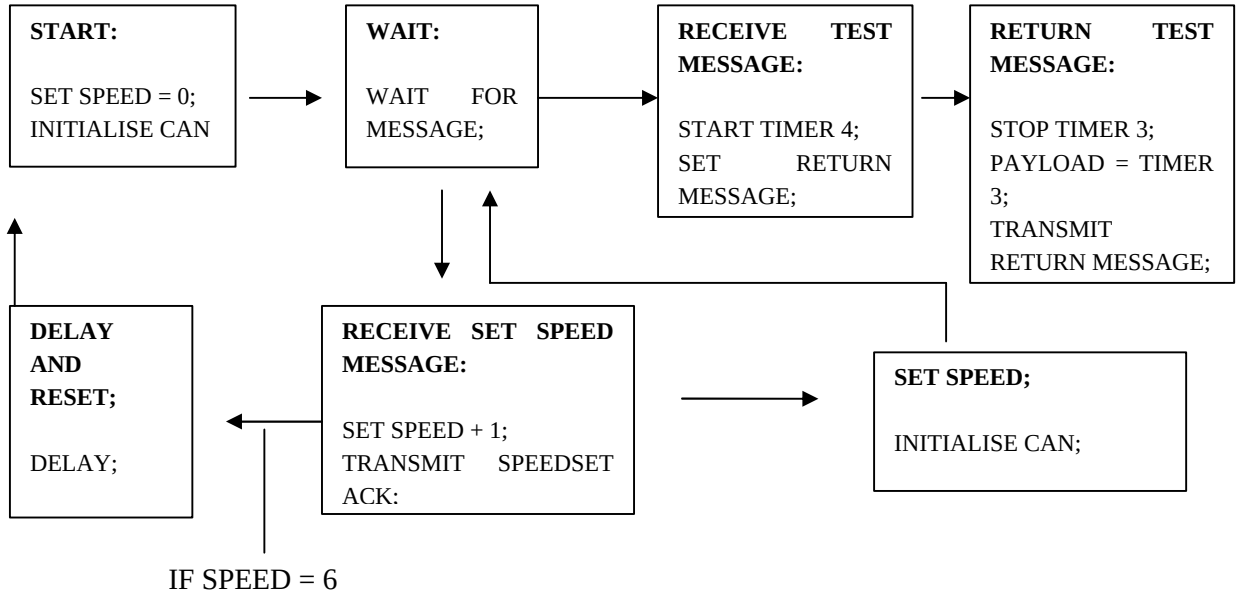


Figure 5.3: Finite State Machine of the Sender

Remarks:

AD **START**:
Speed 0 = 75 kbps.
Speed 1 = 125 kbps.
Speed 2 = 250 kbps.
Speed 3 = 500 kbps.
Speed 4 = 800 kbps.
Speed 5 = 1000 kbps.

Ad **START**: Timer 3 runs at a speed of 1.25 Mhz, T = 800 ns.

Ad **TRANSMIT TEST MESSAGE**: The transmitter test message had a ID 003, which will be recognised as test message by the receiver.

Ad **RECEIVE TEST MESSAGE: SAVE RESULT**: saves value of T3 at the moment of reception (this is the looptime) and the receiver CPU time, which is send in the payload of the returned testmessage. See also finite state machine receiver.

Ad **SET SPEED + 1**:
The speed set message has a specific payload: the first byte of the payload has the value 0x0f, which is recognised by the receiver as a 'increase speed' command.

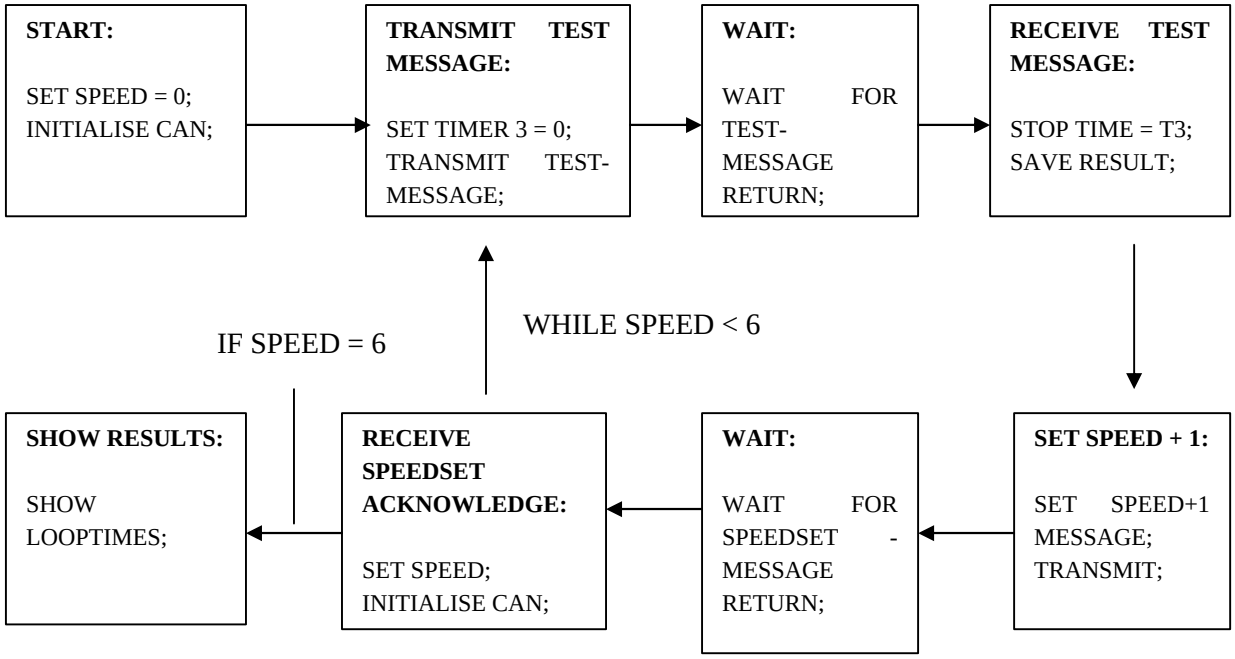


Figure 5.4: Finite State machine of the receiver

Remarks:

AD **START**:
Speed 0 = 75 kbps.
Speed 1 = 125 kbps.
Speed 2 = 250 kbps.
Speed 3 = 500 kbps.
Speed 4 = 800 kbps.
Speed 5 = 1000 kbps.

Ad **RECEIVE TEST MESSAGE:**
Timer 3 runs at a speed of 1.25 Mhz, T = 800 ns.

Ad **RETURN TEST MESSAGE:**
The receiver returns the test message with an ID 222, which will be recognised as test message by the sender.

Ad **DELAY**:
The delay is set for about 1 second.

Used hardware:
Sender and receiver: I+ME Actia C167CD Development Board
EGSTON power supply 12 V DC , 500 mA

Used Software:
C-code compiler: Tasking DemoTools for C166/ST10 v7.5 r2.
HEXLOAD 1.20 © 1994 S. Huehne

The used software is Tasking Demo Tools. In chapter 4, the preferred software was described to be Keil. At first the Keil compiler has been used as compiler for the microcontroller. Unfortunately the debugger did not support hardware debugging on the IME ACTIA CAN boards. The Tasking debugger did work on the boards and therefore became the preferred software tool.

Calculating the Looptime

The results putted out on the RS232 are times in microseconds. These times consist of three parts:
- transfer time of the test-message between sender and receiver
- cpu time of the receiver
- transfer time of the test-message between receiver and sender.

The processing time of the receiver is sent in the payload of the returning message. The initial idea was to put this time on the RS232 port of the receiving CAN board. Since it isn't possible to trace both serial port of the pc with the Tasking DemoTools and another program wasn't available, the choice had been made to put the CPU time of the receiver as payload in the message from receiver to sender. In this way the sender receives the CPU time of the receiver by a CAN message.

The measurement done can be shown by a timing diagram, found in fig 5.5.

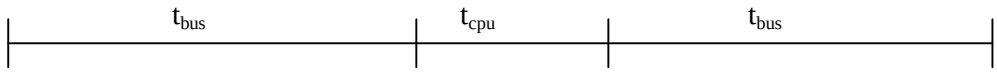


Figure 5.5: timing diagram of the looptime

This diagram can be translated into a formula for the time of the time it takes a message to be transferred from sender to receiver or vice versa. This formula is given in equation 5.1

$$t_{bus} = \frac{t_{total} - t_{cpu,receiver}}{2} \text{ (Eq 5.1)}$$

Unfortunately the value for $t_{cpu,receiver}$ cannot be calculated exactly.

The error made in the processing time is due to the instructions done by the microcontroller after the moment timer 3 is stopped. To get a better view on the instructions, the source code is copied in the next part:

```
void Receive (void)
{
if (C167CInterrupt == 0x03)          /* Rx Int from Msg 1 & Check for ID 3*/
{
        T3=0;
        C167CMessage1 [1] = 0xFD;          /* Reset "NewDat" bit */
        C167CMessage1 [0] = 0x99;          /* Reset "IntPnd" bit */
        bounce_value = C167CMessage1 [7];
        Bounce();                          /*Bounce payload */
}
}

void Bounce (void)
{
C167CMessage2 [0] = 0x55;                  /* setup transmitter mailbox */
```

```
C167CMessage2 [2] = 0x1B;                /* ID 222                */
C167CMessage2 [3] = 0xC0;
bounce_value = T3 ;
C167CMessage2 [7] = bounce_value;  /* put data: bounce received data */
C167CMessage2 [0] = 0x95;                /* activate mailbox      */
C167CMessage2 [1] = 0xE5;                /* transmit request   */
}
```

In the part above, the underlined text is the comment to the code. The italic words are the commands for setting the timer for calculating the processing time. The bold text contents the commands to be processed after the timer is stopped. The time to execute these commands should be added to the processing time putted out by the receiver. To get a value for this execution time, the assembly code is studied. For each command four lines of assembly code is generated. The CPU time for executing this assembly has been set to an average of 3 clock times of the cpu. Unfortunately the datasheets of the microcontroller couldn't give the right value. The estimated time to execute the blue commands can be calculated:

$$3\times3\times3\times \frac{1}{20\cdot10^{6}} = 0,45ms$$

Since the CPU time is given in terms of 1 µs, the CPU time can be set to have an error of ± 1 µs.

5.1.2 Point-to-point results: measurements

As in the previous section has been told, the point-to-point connection is measured at six speeds, 75, 125, 250,500, 800 and 1000 kbps. At each of these speeds the looptime of a message is measured. The measuring consists of the total looptime and the CPU time at the receiver. The measuring of the loop and CPU time is repeated at least 10 times. These measurements are done for payloads from 1 byte to 8 bytes. The results of the point to point connection on the CAN bus can be found in Appendix 2. Since there are a lot of measurements per payload, the averages are taken and displayed in table 5.1.

	Time @ 75 kb/s (µs)	**CPU Time** (µs)	**Time @125 kb/s** (µs)	**CPU Time** (µs)	**Time @ 250 kb/s** (µs)	**CPU Time** (µs)	**Time @ 500 kb/s** (µs)	**CPU Time** (µs)	**Time @ 800 kb/s** (µs)	**CPU Time** (µs)	**Time @ 1000 kb/s** (µs)	**CPU Time** (µs)
Payload 1 byte	2370 ± 3	11 ± 1	1455 ± 3	10 ± 1	748 ± 3	11 ± 1	400 ± 3	11 ± 1	276 ± 3	11 ± 1	221 ± 3	10 ± 1
Payload 2 bytes	2752 ± 3	11 ± 1	1697± 3	11 ± 1	873 ± 3	10 ± 1	457 ± 3	11 ± 1	313 ± 3	11 ± 1	255 ± 3	10 ± 1
Payload 3 bytes	3062 ± 3	10 ± 1	1873 ± 3	11 ± 1	955 ± 3	10 ± 1	502 ± 3	11 ± 1	345 ± 3	11 ± 1	277 ± 3	10 ± 1
Payload 4 bytes	3363 ± 3	11 ± 1	2075 ± 3	11 ± 1	1068 ± 3	11 ± 1	554 ± 3	11 ± 1	383 ± 3	11 ± 1	306 ± 3	11 ± 1
Payload 5 bytes	3757 ± 3	10 ± 1	2285 ± 3	10 ± 1	1165 ± 3	11 ± 1	612 ± 3	11 ± 1	418 ± 3	10 ± 1	336 ± 3	10 ± 1
Payload 6 bytes	3846 ± 3	11 ± 1	2362 ± 3	11 ± 1	1208 ± 3	11 ± 1	643 ± 3	10 ± 1	432 ± 3	11 ± 1	346 ± 3	11 ± 1
Payload 7 bytes	4379 ± 3	11 ± 1	2662 ± 3	11 ± 1	1381 ± 3	11 ± 1	716 ± 3	11 ± 1	482 ± 3	11 ± 1	389 ± 3	11 ± 1
Payload 8 bytes	4699 ± 3	11 ± 1	2922 ± 3	10 ± 1	1500 ± 3	10 ± 1	770 ± 3	11 ± 1	526 ± 3	11 ± 1	422 ± 3	11 ± 1

Table 5.1: Looptime measurement results

From these numbers a looptime, speed graph can be made, which can be found in figure 6.1 below. An extended version can be found in Appendix 3.

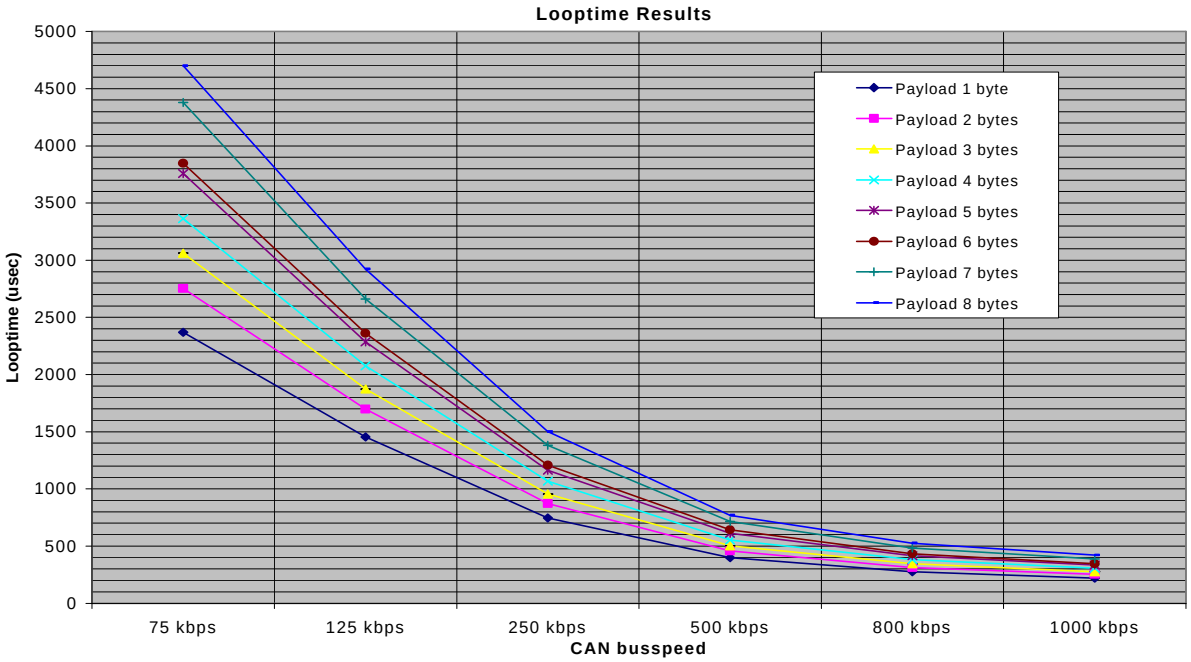


Figure 5.6 Looptime versus CAN bus speed at payload from 1 byte to 8 bytes.

5.1.3 Point-to-point connection: discussion

Introduction

This section will discuss the results of the point-to-point connection.

Efficiency

From the measured results it is possible to calculate a value for the effective transfer speed. This value indicates the best efficiency between data transfer and speed. Since each transfer has protocol overhead the effective bus speed is not equal to actual bus speed. To calculate the efficiency, formula 5.2 is used.

$$h = \frac{2 \cdot 8 \cdot (\#bytes)}{\frac{t_{loop} - t_{cpu}}{1000}} \quad (5.2)$$

In formula 5.2 # bytes is equal to 7 protocol bytes + the number of payload bytes.
This formula is calculated for the values of 1, 4 and 8 bytes payload. The results of the calculation can be found in table 5.2.

Speed (kb/s)	**75**	**125**	**250**	**500**	**800**	***1000***
$V_{bus,eff}$ 1 byte payload (kb/s)	54	89	174	329	483	*607*
$V_{bus,eff}$ 1 byte payload (%)	72 %	71 %	69 %	66 %	60 %	*61 %*
$V_{bus,eff}$ 4 bytes payload (kb/s)	53	85	167	324	473	*597*
$V_{bus,eff}$ 4 bytes payload (%)	70 %	68 %	67 %	65 %	59 %	*60 %*
$V_{bus,eff}$ 8 bytes payload (kb/s)	55	88	172	337	497	*623*
$V_{bus,eff}$ 8 bytes payload (%)	*73 %*	*70 %*	*69 %*	*67 %*	*62 %*	*62 %*

Table 5.2: effective speed calculated for1, 4 and 8 bytes payload

From the results of table 5.2 a graph can be made. This graph can be found in figure 5.7.

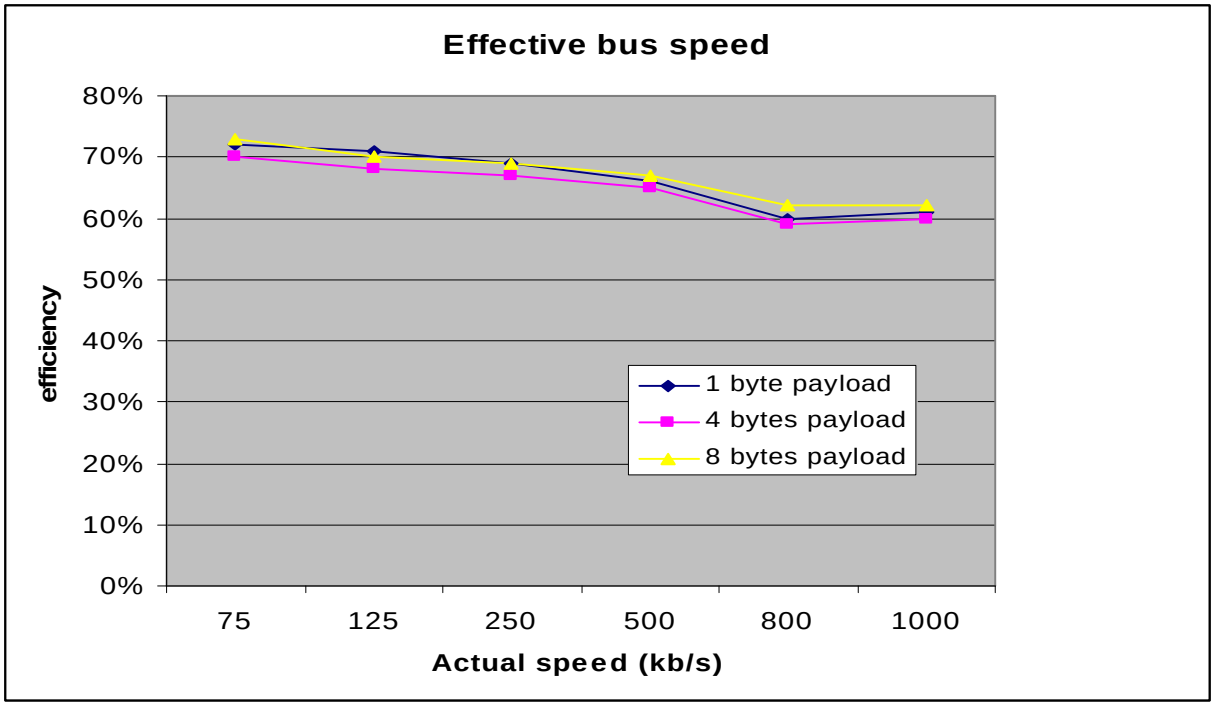


Figure 5.7: effective bus speed

Another interesting calculation is the relation between looptime and payload size. From the results in table 5.1 it can be seen that the looptime increase with the increase of payload. To calculate the relation between the looptime and payload size, the 'single way' time is calculated. This means that the time of transfer between sender and receiver (or vice versa) is used. From the available looptime results this time can be calculated with equation 5.1.

The results of the calculation can be found in table 5.3 and figure 5.8.

Payload	**1 byte**	**2 bytes**	**3 bytes**	**4 bytes**	**5 bytes**	**6 bytes**	**7 bytes**	**8 bytes**
Time @ 75 kb/s (µs)	1180	1371	1526	1676	1874	1918	2184	2344
Time @ 125 kb/s (µs)	722	843	931	1032	1138	1176	1326	1456
Time @ 250 kb/s (µs)	369	431	473	529	577	599	685	745
Time @ 500 kb/s (µs)	195	223	246	271	301	317	353	380
Time @ 800 kb/s (µs)	132	151	167	186	204	211	236	258
Time @ 1000 kb/s (µs)	105	122	134	148	163	167	189	206

Table 5.3: single way times at various payloads

As can be seen form figure 5.8, each speed has a certain offset, when the line in extrapolated to the 0 byte payload. This is caused by the protocol overhead.

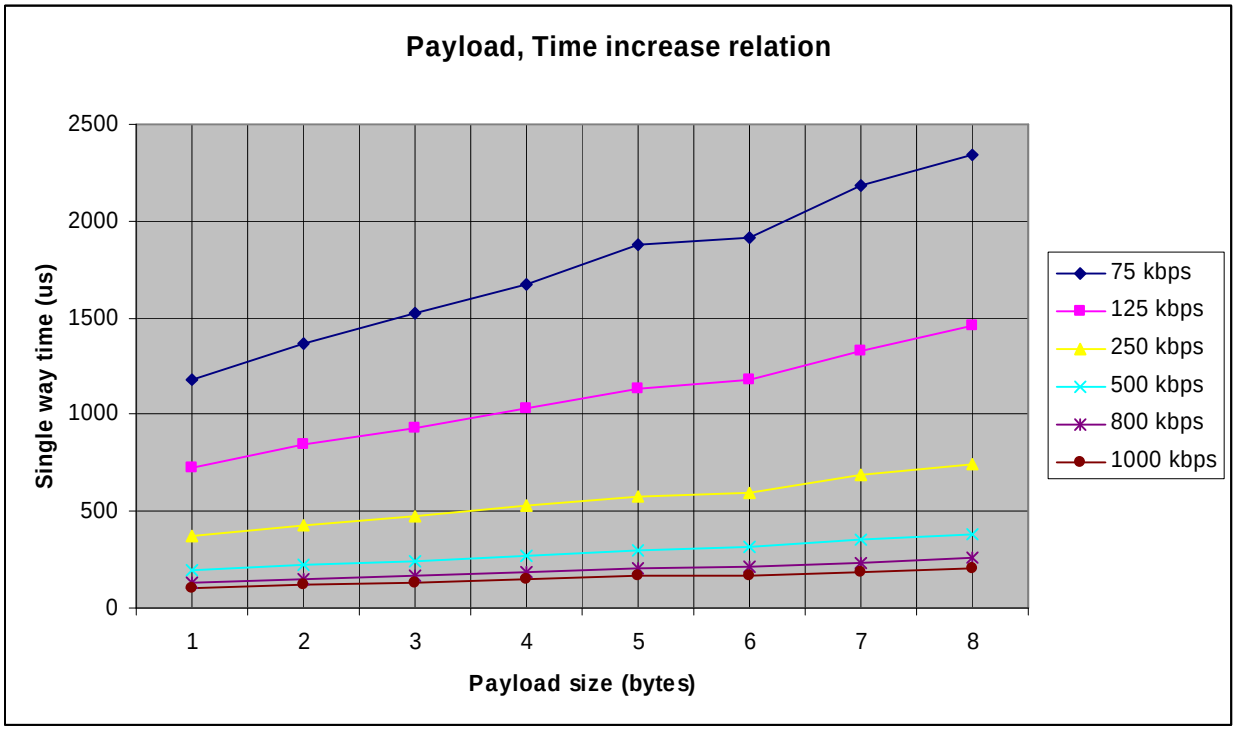


Figure 5.8: Payload, Time increase relation

5.2 Arbitration

5.2.1 Arbitration: measurement setup

The second measurement is about arbitration. Since CAN is a bus, the transfer of messages is not always possible. If more than 2 nodes (say machines) are connected to the bus, there are moments the bus is busy. If so, there will be a delay in transferring messages over the bus. Since in real-time control this time cannot be very long, preferring zero, we must have a closer view on this problem.

CAN uses a protocol that deals with bus control. This is called arbitration. With arbitration it is possible to have more important messages that are more time critical. The concept is that each node can give 14 levels of priority to transfer of messages. At an equal rate of messages to be sent and to be received, one may say that there are 7 levels of priority for sending messages and 7 levels of priority in receiving messages. The goal of this second measurement is to get a clear view of the arbitration principal on CAN. Questions one may ask are:

- How the arbitration does work on sending and receiving messages, are both ways the same?
- Can there be a timing guarantee for high priority messages / real time control?
- What is the relation between timing guarantees and the messages configuration (payload, speed)?

This part will give the overview of the arbitration principal measurement. Arbitration is applied for sending messages at the same time. First the focus will be on sending messages. The measurement of the send-arbitration is based on the looptime measurement. At a fixed speed and fixed payload the sender sends four messages at a time to the receiver. All the four messages have a different priority. A third node (CAN card) will monitor the bus activity. Figure 5.9 gives the block diagram. The priority will be varied among the four messages. The goal is to proof that the message with the lowest priority number (which indicates that the message has the highest priority) is send first.

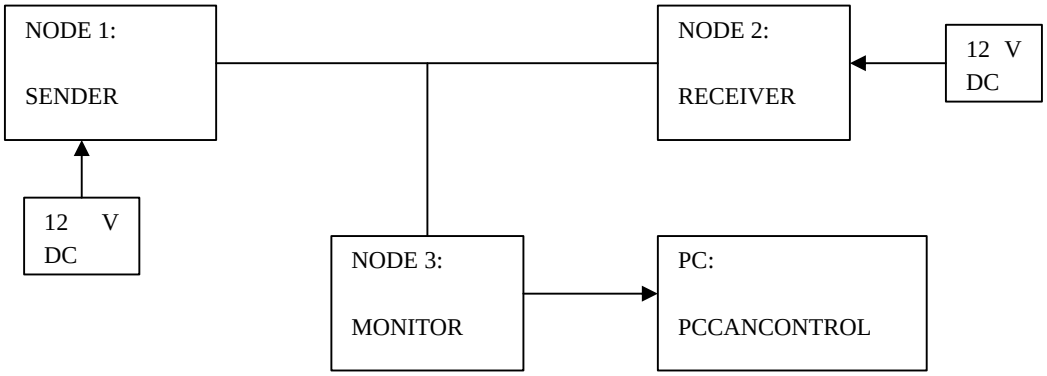


Figure 5.9: Block diagram Send arbitration

Used Hardware:
Node 1 & 2: I+ME ACTIA C167CR Development boards
Node 3: I+ME ACTIA PcSlimLine ISA Card
EGSTON power supply 12 V DC , 500 mA

Used Software:
C-code compiler: Tasking DemoTools for C166/ST10 v7.5 r2.
HEXLOAD 1.20 © 1994 S. Huehne
Monitoring: PC CAN/ISO Control for Windows 1.06

5.2.2 Arbitration: results

The send arbitration is measured with PCCANCONTROL as described in paragraph 5.2.1. By checking which message (with a certain priority) is send first on the bus, the send arbitration is pointed out. The best way to explain the functionality is by figure 5.10, which is the result in PCCANCONTROL. In figure 5.10 the messages send over the bus are shown. In the left column the timestamps generated by PCCANCONTROL are shown. In the next column the identifiers can be found as programmed by the c-code. The column Type, Len and Data show the type of the message (standard or extended CAN), the payload length in bytes and the data of the payload. Until timestamp 367377.400 the identifier of the message is the same as the priority level of the message. Here the three test messages have identifiers and thus priority 4, 6 and 8. Theoretically the message with the highest priority, with the lowest priority*number* should be sent first. In the figure it is shown that the bus controller sends them in this way. The message with identifier 3 is send before sending the 3 messages at the same time. This message is a check for functionality and has nothing to do with the send arbitration measurement.

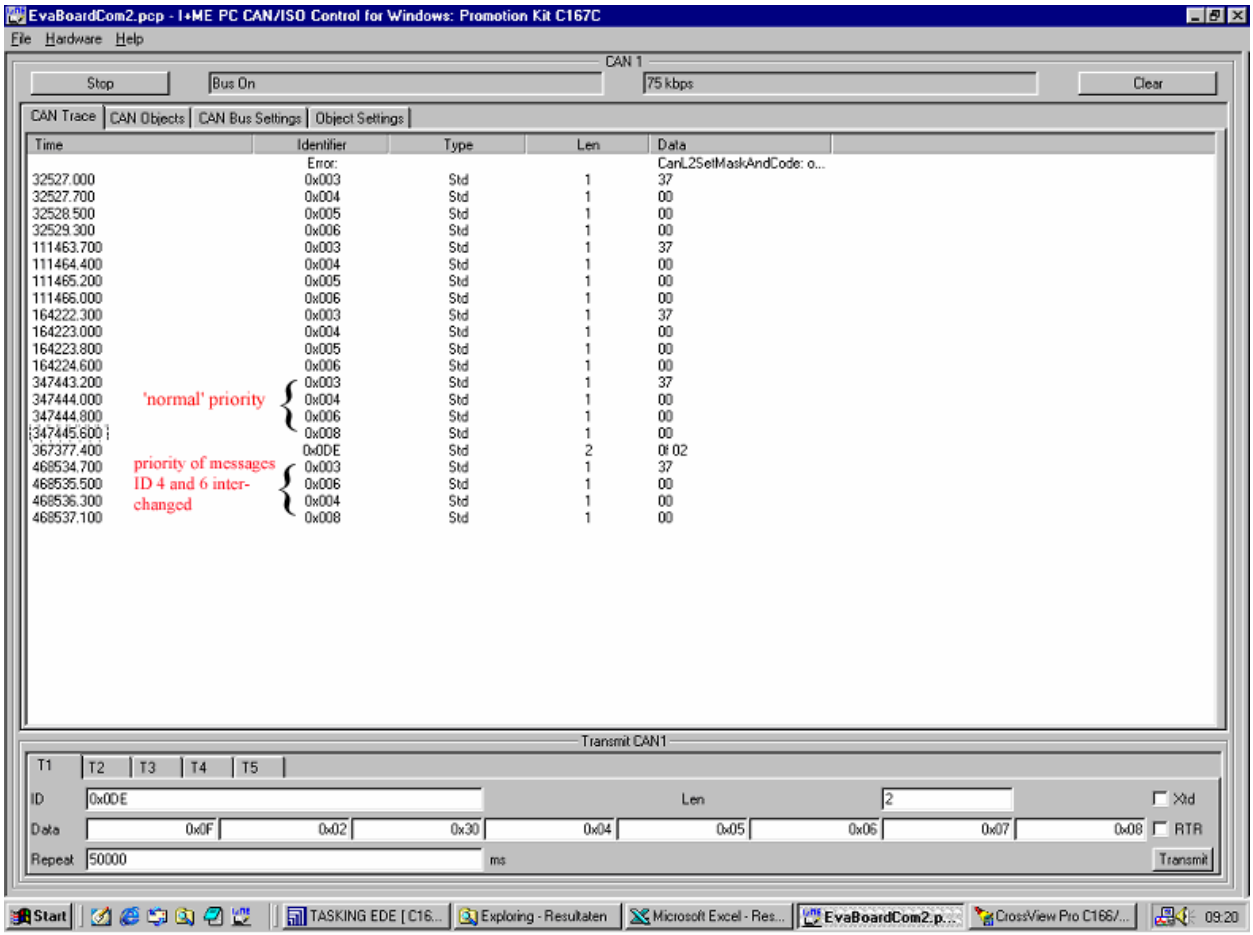


Figure 5.10: PCCANCONTROL

5.2.3 Arbitration: discussion

After timestamp 367377.400 the messages with identifier 4 and 6 are switched prioritynumber. The consequence should be that the moment of sending these two messages should switch also. As we can see at timestamps 468535.500 and 468536.300 this is the case.

5.3 Busload influence

Introduction

This section will discuss the final measurement of the bus influence.

5.3.1 Busload influence: measurement setup

The last part of the measurements is to get a view on how the arbitration principal works with various busloads. For this a third node will be introduced to the network. This third node has the function of loading the CAN bus, by sending messages. With PCCANCONTROL it is possible to let the PC SLIMLINE card send messages with payloads varying between 1 and 8 bytes and send it at different bus speeds. The maximum send frequency is 1000 Hz. Unfortunately this frequency is a little low when using the CAN bus speed of 500 kbps and higher. The goal of this measurement is to get insight in the influence of the busload on sending highest priority messages The third node will send messages to fill the bus and a series of 25 looptime messages will be send on the bus. The looptimes of these messages will be measured and compared to the point-to-point connection (this means a busload of 0%).

The questions posed in section 5.2.1 will also be answered by using the looptime measurement. By applying a third node to the network, it is possible to apply more bus traffic and thereby make the CAN bus busy. The block diagram of this network can be found in figure 5.11.

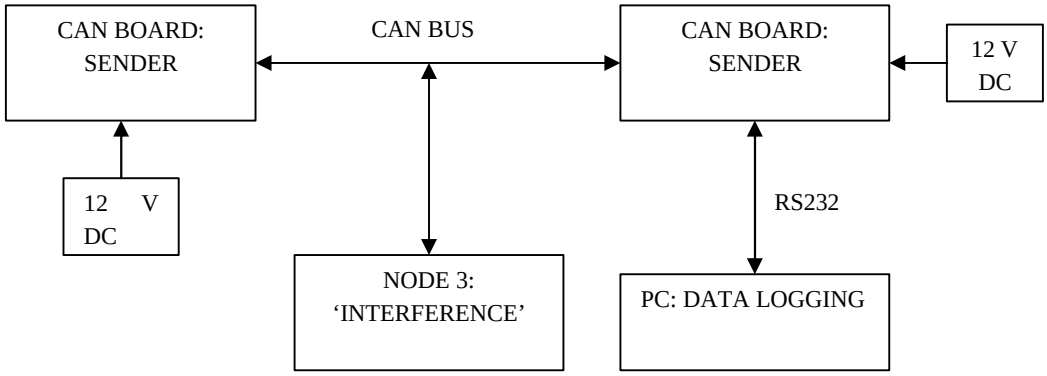


Fig. 5.11: Block diagram interference

The influence of this busy state on the looptime will be measured. Also different configurations of bus and messages will be applied with this measurement.

Used Hardware:
Node 1 & 2: I+ME ACTIA C167CR Development boards
Node 3: I+ME ACTIA PcSlimLine ISA Card
EGSTON power supply 12 V DC , 500 mA

Used Software:
C-code compiler: Tasking DemoTools for C166/ST10 v7.5 r2.
HEXLOAD 1.20 © 1994 S. Huehne
Monitoring: PC CAN/ISO Control for Windows 1.06

5.3.2 Busload influence: Results

As described in the previous section a set of 25 looptime measurements is done e at various speeds with a fixed payload of 8 bytes. For speeds lower than 500 kbps the number of measurements is set to 10. The reason for this is an overflow in the variable msec_temp, which is the variable for the total time the measurements take. At lower speeds the looptime of 25 measurements was too big to show correctly.

With PCCANCONTROL it is possible to send messages at a maximum frequency of 1 kHz. Unfortunately this value is rather low for high bus speeds. It is therefore not possible to 'fill' the bus completely at high speed connections (250 kbps and up). The measurements done will be discussed now.
In table 5.4 and 5.5 the results of 25 looptime measurements are shown at bus speeds of 500kbps and 1 Mbps. In table 5.6 and 5.7 the results of 10 looptime measurements are shown at bus speeds of 250 kbps and 125 kbps.

The busloads T1 through T5 shown in the tables refer to the 5 channels at which PCCANCONTROL can send messages. It is possible to send 5 different messages with the program at the same time. The reason to use more than one channel (for example only T1) is to fill the bus with more data, since the send frequency used in PCCANCONTROL is rather low.

Measurement	Time with no busload (µs)	Time with busload T1 (µs)	Time with busload T1,T2 (µs)	Time with busload T1,T2, T3 (µs)	Time with busload T1 through T4 (µs)	Time with busload T1 through T5 (µs)
1	19391	19391	19396	**20153**	19397	19396
2	19394	19392	19398	19393	**20659**	19389
3	19389	**19717**	**20026**	19392	19394	19393
4	19389	19394	19387	19392	19396	**20972**
5	19393	19390	19389	**20139**	19390	19397
6	19394	19394	19399	19395	**20661**	19394
7	19391	**19718**	**20028**	19401	19394	19392
8	19391	19393	19393	19395	19360	**20976**
9	19393	19395	19396	19394	19395	19396
10	19393	19395	19392	19390	**20662**	19393

Table 5.4: results of 25 looptime measurements at 500 kbps CANspeed

Measurement	Time with no busload (µs)	Time with busload T1 (µs)	Time with busload T1,T2 (µs)	Time with busload T1,T2, T3 (µs)	Time with busload T1 through T4 (µs)	Time with busload T1 through T5 (µs)
1	10551	10563	10559	10545	10555	10553
2	10551	**10703**	**10825**	10558	10555	10558
3	10546	10562	10554	10551	10545	10548
4	10550	10556	10555	10551	10558	10553
5	10552	10562	10539	10558	10547	**11188**
6	10549	10566	10554	10545	10549	10553
7	10555	10549	10551	**10985**	**11156**	10560
8	10556	10554	10547	10564	10558	10550
9	10564	**10686**	10556	10560	10549	10544
10	10559	10557	10569	10557	10558	10560

Table 5.5: results of 25 looptime measurements at 1Mbps CANspeed

Measurement	Time with no busload (µs)	Time with busload T1 (µs)	Time with busload T1,T2 (µs)	Time with busload T1,T2, T3 (µs)	Time with busload T1 through T4 (µs)	Time with busload T1 through T5 (µs)
1	14953	14949	14949	14950	14950	14951
2	14947	14947	14948	**16648**	14947	14946
3	14951	14948	14947	14952	14950	14950
4	14957	**15621**	14947	14947	**17290**	**18331**
5	14953	14948	**16299**	14951	14953	14949
6	14949	14948	14950	14946	14950	14949
7	14953	14945	14948	**16976**	14953	14952
8	14948	14944	14948	14950	14951	14950
9	14950	14948	14949	14954	**17649**	**18316**
10	14954	**15621**	**16299**	14946	14948	14944

Table 5.6: results of 10 looptime measurements at 250 kbps CANspeed

Measurement	Time with no busload (µs)	Time with busload T1 (µs)	Time with busload T1,T2 (µs)	Time with busload T1,T2, T3 (µs)
1	29067	29065	29065	29067
2	29064	**30474**	29066	29060
3	29064	29059	**31897**	**overflow**
4	29076	**30443**	29065	29057
5	29077	29068	29063	29060
6	29070	29066	**30441**	**31845**
7	29068	**30481**	29066	29064
8	29067	29063	**31896**	**overflow**
9	29064	29061	29062	29069
10	29064	**30489**	29057	29059

Table 5.7: results of 10 looptime measurements at 125 kbps CANspeed

5.3.3 Busload influence: discussion

Of the results of tables 5.4 through 5.7 it can be seen that the time for the 10 or 25 looptime measurements increases due to the busload. The results of these calculations can be found in table 5.8 through 5.11.

	Busload 0	Busload T1	Busload T1,T2	Busload T1,T2,T3	Busload T1 through T4	Busload T1 through T5
Average time (µs)	19392	19458	19520	19544	19771	19710
Maximum difference (µs)	0	326	636	761	1270	1584
Relative difference	0	1,68%	3,28%	3,93%	6,55%	8,17%

Table 5.8: Average looptime and time difference for bus speed 500 kbps

	Busload 0	Busload T1	Busload T1,T2	Busload T1,T2,T3	Busload T1 through T4	Busload T1 through T5
Average time (µs)	10553	10586	10581	10597	10613	10617
Maximum difference (µs))	0	150	272	432	603	635
Relative difference		1,42%	2,57%	4,09%	5,71%	6,01%

Table 5.9: Average looptime and time difference for bus speed 1000 kbps

	Busload 0	Busload T1	Busload T1,T2	Busload T1,T2,T3	Busload T1 through T4	Busload T1 through T5
Average time (µs)	14952	15082	15218	15322	15454	15624
Maximum difference (µs)	0	670	1348	2025	2698	3380
Relative difference	0	1,68%	3,28%	3,93%	6,55%	8,17%

Table 5.10: Average looptime and time difference for bus speed 250 kbps

	Busload 0	Busload T1	Busload T1,T2
Average time (µs)	29068	29627	29768
Maximum difference (µs)	0	1421	2829
Relative difference	0	4,89%	9,73%

Table 5.11: Average looptime and time difference for bus speed 125 kbps

These results can be summarised by figure 5.12. In this figure it is clear that the delay in looptime increases when loading the bus with more messages.

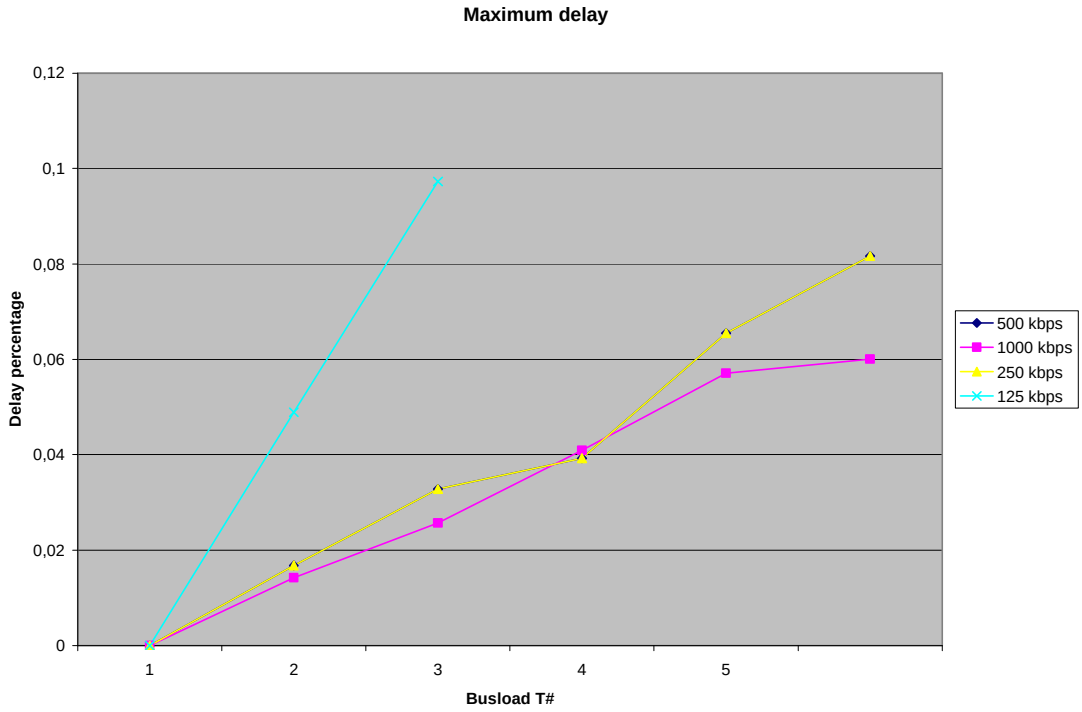


Figure 5.12: Maximum delay

Now the three measurements are completely discussed, it is time to draw conclusions. This is done in the next chapter.

6. Conclusions

Introduction
Now all tests have been done, the conclusions of the results can be drawn.

6.1 Conclusions

From the results of the point-to-point measurement the following conclusions can be drawn.

1. The looptime decreases when the bus speed increases. The relation between looptime and bus speed is linear.
2. The effective bus speed, this means the actual data transfer speed, is about 60 to 70 % of the set bus speed. This results in a maximum data transfer speed on CAN of 600 to 700 kbps at a bus speed of 1 Mbps. (which is the CAN speed)
3. The CAN protocol uses 14 levels of priority. The message with the highest priority wins arbitration in case when 2 or more messages are sent at the same time.
4. When the bus is busy, the CAN controller holds the message to be sent, even when it has higher priority. In this way messages don't get lost, but holding the message results in a delay.
5. In the three node network 20 % of the data transfer is affected by the caused busload (8 bytes payload messages send at 1 kHz). At speeds of 250 kbps and lower this percentage increases to 40 %.
6. In the three node network the delay is about 5 % of the total transfer time, with the caused busload. (8 bytes payload messages send at 1 kHz)

6.2 Recommendations

This section points out some possibilities to continue the research with the CAN development tools as are available now.

To make bigger and more complex applications with the CAN development boards it is necessary to use a registered version of the Tasking compiler.

To results of the measurements show delays in data transfer when using more than two nodes in the network. These delays are bad for real time control. Therefore the functionality of time-triggered CAN should be explored. This is possible with the available hardware. One of the development boards can serve as the time master. By defining timeslots for each task to be performed, it is possible to use the bus more efficiently and delays can be minimised.

7 Appendices

7.1 USB 2 Report

1 Real-time aspects focused on USB 2

Introduction

In the previous chapters various aspects of real-time communication were mentioned. This final chapter will analyze USB 2 by reviewing these aspects focused on USB 2. The goal of this chapter is conclude whether USB 2 is suitable for real-time communication.

protocol latency

The commands used in the USB are generated by the host. Due to this hierarchy in the USB protocol there is some protocol latency, which can be distinguished in:

- State handling: the monitoring of states of devices attached to the USB
- Serializer/deserializer: the datastream transferred over the USB is send in a serial bitstream. The serial interface engine (SIE) is located at the host or partly at the device. The SIE adds some protocol latency during serializing the data.
- Frame / Microframe generation: Since data over the USB is send in frames (microframes at high speed), the generation of these frames adds some protocol latency.

Composability

The USB is controlled by the host. The host sends data over the bus. The host can be seen as external control on the bus and therefore some of the nodes connected to the bus can be reserved for a specific operation. For this reason USB can make use of composability.

Flexibility

The USB supports various transfer modes. For each mode a specific bandwidth can be reserved.

Flow control

Each transfer is started with a handshake. Each transfer of data has to be acknowledged by the endpoint it is send to. In this way the data on the USB is under explicit flow control.

2. Introduction to USB 2.0

Introduction

This part of the report focuses on the USB (Universal Serial Bus) 2.0. The goal of this report is to give a short and clear overview of the overall working of the bus. The main source to this report is the datasheets of the USB 2.0 [USB.org, 2000 #8]

The USB is specified to be an industry-standard extension to the PC architecture with a focus on PC peripherals that enable consumer and business applications. The following criteria were applied in defining the architecture for the USB:

- Ease-of-use for PC peripheral expansion.
- Low-cost solution that supports transfer rates up to 480 Mb/s.
- Full support for real-time data for voice, audio, and video.
- Protocol flexibility for mixed-mode isochronous data transfers and asynchronous messaging.
- Integration in commodity device technology.

- Comprehension of various PC configurations and form factors.
- Provision of a standard interface capable of quick diffusion into product.
- Enabling new classes of devices that augment the PC's capability.
- Full backward compatibility of USB 2.0 for devices built to previous versions of the specification.

The first part of the report focuses on the hardware in the bus. Afterwards the dataflow (transfers and protocols) will be focused on.

2.1 Bus topology

Introduction

The bus topology is organized in a star-formed chain. The center of each star is the host. In each chain either a hub or a USB device forms the link. Due to timing constraints and cable propagation times a maximum number of 7 devices may be connected to a hub. So the chain in the topology can only consist of 7 levels. This topology can be schematically represented by figure 5.1.

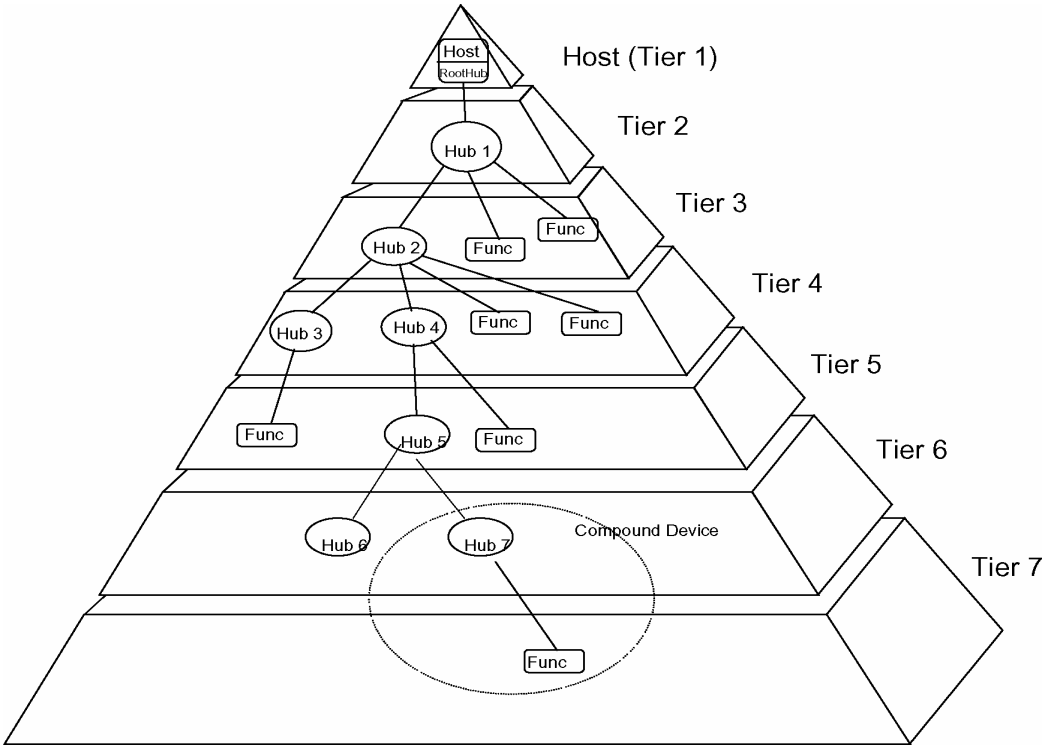


Figure 2.1: bus topology of USB

As can be seen in figure 2.1 the host is connected to a maximum of 7 levels of hosts and devices (which are named 'Func' in the figure).

2.2 Hardware

The hardware of USB consists of three parts:

1) Host
2) Interconnect
3) HUB/Device

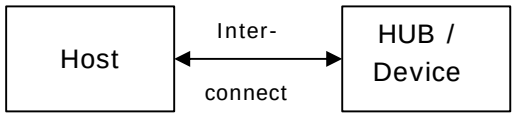


Figure 2.2: schematical overview of USB

The relationship between the three parts has been pointed out in figure 2.2.

The Host

The host provides the communication steering between the client software and the functionality of the USB device. The host monitors attachment and removal of USB devices, manages the control flow in the bus and monitors the activity status of the bus and attached devices. The host also provides the devices with power.

The interconnect

The interconnect is the cable between the host and device or hub attached. The schematic view of the cable is shown in figure 2.3:

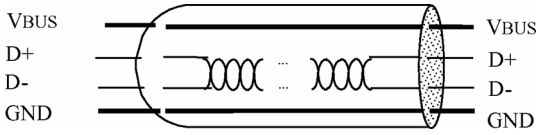


Figure 2.3: Schematic overview of the USB cable

The D+ and D- signals are the data signals. These lines contain a NRZI (Non Return to Zero Invert) signal. The V_{bus} and GND provide the hub and/or USB device of power. The cables used in USB 1.x can also be used in USB 2.0.

HUB

A USB HUB has generally three functions. The first is the one of repeater. The HUB controls up and downstream of data signals, reset, suspend and resume commands. Secondly, the HUB controls the communications to and from the host. Finally the HUB acts like a data-speed controller. The communications between Host and Hub is in *high speed* and the communications between HUB and device can be on *full or low speed*. The various speeds mentioned here will be explained in chapter 6 ‘ Data flows’

USB Device

Three aspects characterise a USB device: the bus interface, the logical device and the function. The last aspect, the function, is most important one. In the web of hubs and devices, each USB device adds some functionality. Therefore the USB device can be seen as a function rather than a device. It is also possible to have more functions in one device. For example, a keyboard with trackball has two functions: the keyboard function and the trackball function. Within the device a hub interconnects the two functions.

3. Data flows

Introduction

This chapter will now focus on a more detailed level at the communication lines within the USB. First the various flows will be mentioned and after that the type of transfers will be focused on.

Speed

The communications transfer rates within USB 2.0 are capable of three speeds:

- Low speed: 1,5 Mbit/s
- Full speed: 12 Mbit/s
- High speed: 480 Mbit/s

It depends on the application/device which speed is used. As has been told in the chapter 5, the host is connected to a web of hubs and/or devices. The speeds between the devices and hubs can vary between full and low speed and the speed between host and hub or high-speed device is always high speed.

Communication flows

An overview of the communication on the USB can be given by the layout of figure 3.1.

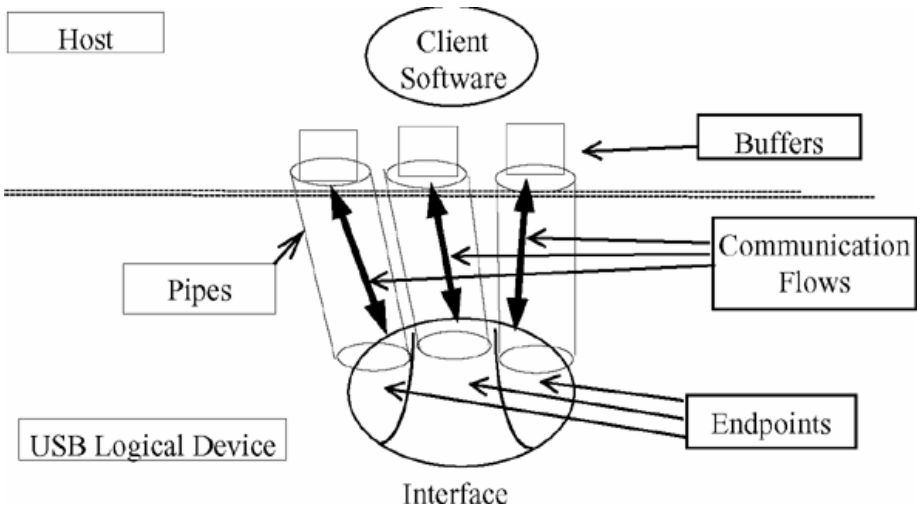


Figure 3.1: Communication flows in USB

Each of the terms used in the layout will be discussed in the following paragraphs.

Endpoints

The communication flow ends at an endpoint in a logical device. Each device is composed of a set of independent endpoints. An endpoint is a simplex connection and can only support data flows in one direction: either input (device to host) or output (host to device). The following aspects describe endpoints:

- Bus access frequency/latency requirement.
- Bandwidth requirement.
- Endpoint number.
- Error handling behavior requirements.
- Maximum packet size that the endpoint is capable of sending or receiving.
- The transfer type for the endpoint (for example: isochronous transfer, bulk transfer, control transfer).
- The direction in which data is transferred between the endpoint and the host

Each USB device has required endpoints, the so-called endpoint zero requirements. These requirements are the ability to reset at full speed and accepting and handling standard requests.

Pipes

A USB pipe is the representation of the association between a device and the software on the host. There are two types of pipes:

- Message pipes: data with some USB defined structure, interacts with endpoints. Data transfer is based on requests. The direction of dataflow can be bi-directional, but not at the same time. The Default Control Pipe, the pipe that is always present after a device is connected, is always a message pipe.
- Stream pipes: data without USB defined structure, unidirectional in data flow.

Frames

USB establishes a 1-millisecond time base called a frame on a full-/low-speed bus and a 125 µs time base called a micro frame on a high-speed bus. A (micro) frame can contain several transactions. Each transfer type defines what transactions are allowed within a (micro) frame for an endpoint. Isochronous and interrupt endpoints are given opportunities to the bus every N (micro) frames. In figure 3.2 a schematical overview is shown

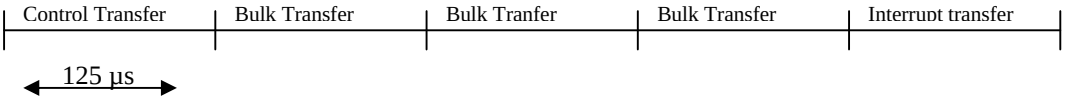


Figure 3.2: Schemtical overview of transfer over USB 2 at high speed

Transfer types

within the data flows in the USB there can be several transfer types distinguished. Each transfer type has typical properties, like:

- Data format imposed by the USB
- Direction of communication flow
- Packet size constraints
- Bus access constraints
- Latency constraints
- Required data sequences
- Error handling

Transfer types

The four transfer types that have been defined for USB are: control transfers

1. Isochronous transfers
2. Interrupt transfers
3. Bulk transfers

Control Transfers

Control transfers allow access to different parts of a device. Control transfers are intended to support configuration/command/status type communication flows between client software and its function. Control transfers are carried only through message pipes. Therefore the data flows using the control transfer must use the USB data structure definitions.

Isochronous transfers

An isochronous transfer is a stream of data whose timing is implied by its delivery rate. Within the USB environment the transfer is provided with a guaranteed bandwidth, a constant data rate and no retrying of deliverance in case of failure. An isochronous transfer is a stream pipe transfer. The USB limits the maximum data payload size to 1,024 bytes for each full-speed isochronous endpoint. High-speed endpoints are allowed up to 1024-byte data payloads.

Interrupt Transfer

The interrupt transfer type is designed to support those devices that need to send or receive data infrequently, but with bounded service periods. Requesting a pipe with an interrupt transfer type provides the requester with the following:

- Guaranteed maximum service period for the pipe.
- Retry of transfer attempts at the next period, in the case of occasional delivery failure due to error on the bus.

High-speed endpoints are allowed maximum data payload sizes up to 1024 bytes.

Bulk Transfers

The bulk transfer type is designed to support devices that need to communicate relatively large amounts of data at highly variable times where the transfer can use any available bandwidth. Requesting a pipe with a bulk transfer type provides the requester with the following:

- Access to the USB on a bandwidth-available basis
- Retry of transfers, in the case of occasional delivery failure due to errors on the bus
- Guaranteed delivery of data but no guarantee of bandwidth or latency

Bulk transfers occur only on a bandwidth-available basis. For a USB with large amounts of free bandwidth, bulk transfers may happen relatively quickly; for a USB with little bandwidth available, bulk transfers may trickle out over a relatively long period of time.

Figure 3.3 will summarise the named transfers. In this figure the schematical build-up of a transfer for each type of transfer is shown. The I/O request packet consists of some control commands (for control transfer) or data-transactions. As can be seen the USB uses packet transfer.

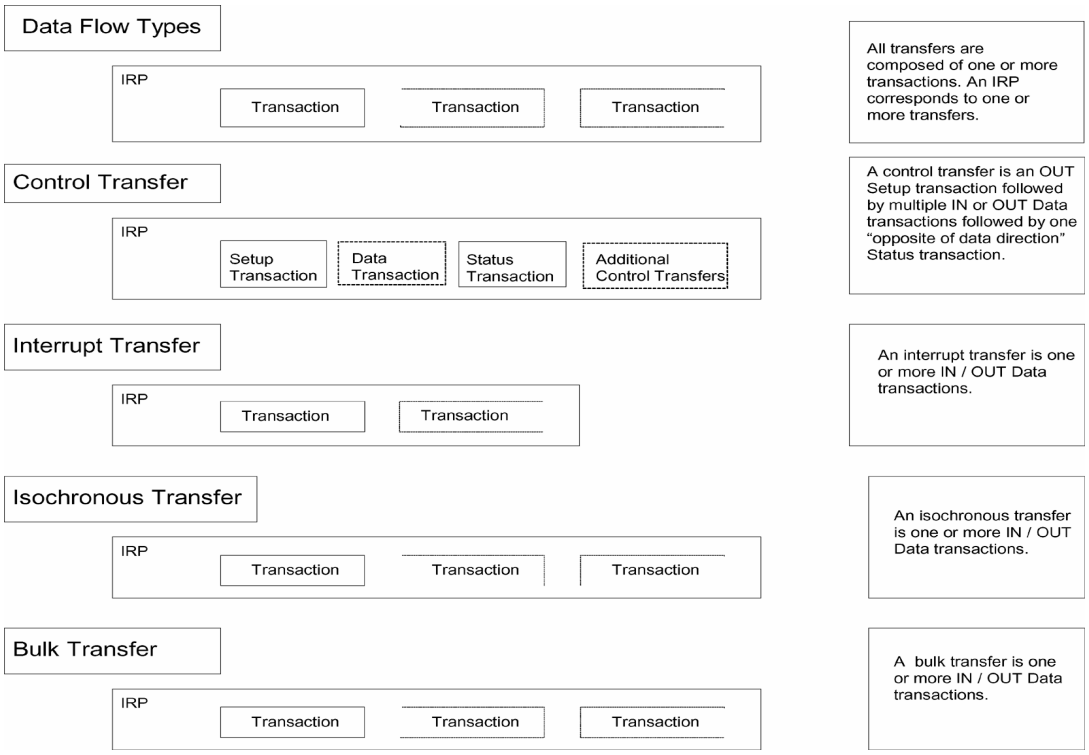


Figure 3.3: Overview of transfer types

Calculation Examples

For each of the transfers a calculated overview will be given in the following figures. In each figure the protocol overhead for high speed (480 Mbit/s) is given, since this is the speed we are most interested in. Each figure shows a table with:

- The protocol overhead required for the specific transfer type at high speed
- For some sample data payload sizes:
    - The maximum sustained bandwidth possible for this case
    - The percentage of a (micro) frame that each transaction requires
    - The maximum number of transactions in a (micro) frame for the specific case
    - The remaining bytes in a (micro) frame that would not be required for the specific case
    - The total number of data bytes transported in a single (micro) frame for the specific case

A transaction of a particular transfer type typically requires multiple packets. The protocol overhead for each transaction includes:

- A SYNC field for each packet: either 8 bits (full-/low-speed) or 32 bits (high-speed)
- A Packet Identifier (PID) byte for each packet: includes PID and PID invert (check) bits
- An End-of Packet (EOP) for each packet: 3 bits (full-/low-speed) or 8 bits (high-speed)
- In a token packet, the endpoint number, device address, and Cyclic Redundancy Check (CRC5) fields (16 bits total)

	Protocol Overhead (173 bytes)		(Based on 480Mb/s and 8 bit interpacket gap, 88 bit min bus turnaround , 32 bit sync, 8 bit EOP: (9x4 SYNC bytes,9 PID bytes,6 EP/ADDR+CRC,6 CRC16, 8 Setup data,9x(1+11) byte interpacket delay (EOP, etc.))			
	Data payload	Max Bandwidth (bytes/second)	Microframe Bandwidth per Transfer	Max Transfers	Bytes Remaining	Bytes/Microframe Useful Data
	1	344,000	2%	43	18	43
	2	672,000	2%	42	150	84
	4	1,344,000	2%	42	66	168
	8	2,624,000	2%	40	79	328
	16	4,992,000	3%	39	129	624
	32	9,216,000	3%	36	120	1152
	64	15,872,000	3%	31	153	1984
MAX		60,000,000				7500

Table 3.4: Transfer limits for Control Transfers

	Protocol Overhead		(Based on 480Mb/s and 8 bit interpacket gap, 88 bit min bus turnaround , 32 bit sync, 8 bit EOP: (2x4 SYNC bytes,2 PID bytes,2 EP/ADDR+addr+CRC5,2 CRC16, and a 2x(1+11)) byte interpacket delay (EOP,etc)			
	Data payload	Max Bandwidth (bytes/second)	Microframe Bandwidth per Transfer	Max Transfers	Bytes Remaining	Bytes/Microframe Useful Data
	1	1,536,000	1%	192	12	192
	2	2,992,000	1%	187	20	374
	4	5,696,000	1%	178	24	712
	8	10,432,000	1%	163	2	1304
	16	17,664,000	1%	138	48	2208
	32	27,392,000	1%	107	10	3424
	64	37,376,000	1%	73	54	4672
	128	46,080,000	2%	45	30	5760
	256	51,200,000	4%	25	150	6400

	512	53,248,000	7%	13	350	6656
	1024	57,344,000	14%	7	66	7168
	2048	49,152,000	28%	3	1242	6144
	3072	49,152,000	41%	2	1280	6144
MAX		60000000				7500

Table 3.5: Transfer limits for Isochronous transfer

Protocol Overhead		(Based on 480Mb/s and 8 bit interpacket gap, 88 bit min bus turnaround , 32 bit sync, 8 bit EOP: (3x4 SYNC bytes,3 PID bytes,2 EP/ADDR+CRC,2 CRC16, and a 3x(1+11)) byte interpacket delay (EOP,etc)			
Data payload	Max Bandwidth (bytes/second)	Microframe Bandwidth per Transfer	Max Transfers	Bytes Remaining	Bytes/Microframe Useful Data
1	1,064,000	1%	133	52	133
2	2,096,000	1%	131	33	262
4	4,064,000	1%	127	7	508
8	7,616,000	1%	119	3	952
16	13,440,000	1%	105	45	1680
32	22,016,000	1%	86	18	2752
64	32,256,000	2%	63	3	4032
128	40,960,000	2%	40	180	5120
256	49,152,000	4%	24	36	6144
512	53,248,000	8%	13	129	6656
1024	49,152,000	14%	6	1026	6144
2048	49,152,000	28%	6	1191	6144
3072	49,152,000	42%	2	1246	6144
MAX	60000000				7500

Table 3.6: Transfer limits for Interrupt transfer

Protocol Overhead (55 bytes)		(3x4 SYNC bytes, 3 PID bytes, 2 EP/ADDR + CRC bytes, 2 CRC16, and a 3x(1+11) byte interpacket delay (EOP, etc)			
Data payload	Max Bandwidth (bytes/second)	Microframe Bandwidth per Transfer	Max Transfers	Bytes Remaining	Bytes/Microframe Useful Data
1	1,536,000	1%	133	52	133
2	2,992,000	1%	131	33	262
4	5,696,000	1%	127	7	508
8	10,432,000	1%	119	3	952
16	17,664,000	1%	105	45	1680
32	27,392,000	1%	86	18	2752
64	37,376,000	2%	63	3	4032
128	46,080,000	2%	40	180	5120
256	51,200,000	4%	24	36	6144
512	53,248,000	8%	13	129	6656
MAX	60000000				7500

Table 3.7: transfer limits for Bulk Transfers

All the tables 3.4 till 3.7 are found at the USB.org website (USB.org,). The tables describe the calculations for various payloads (which are given in kbytes (1 = 1000 bytes)) The number of transfers is calculated by dividing the total bandwidth by the payload. The solution to this division must be divided by 8 since the bandwidth is given in bytes/second and a microframe lasts 125 microseconds.

4. Protocols

Introduction

This chapter will focus on the protocols used within USB2.0.

Byte/bit ordering

The bytes and bits send according to the little Endian method. Figure 4.1 describes this method. This method sends the least significant bits first, followed by the next least significant bit. The last it sent is the most significant bit.

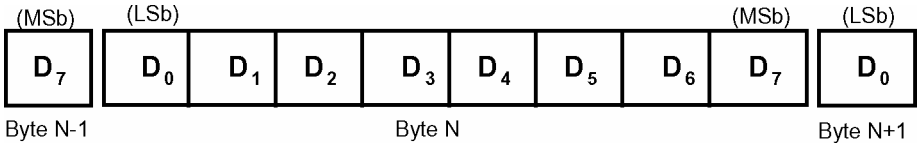


Figure 4.1 Little Endian coding

SYNC field

Each USB packet starts with a SYNC field. This field is 8 bits in full and low speed and contains 32 bits in high speed. The goal of this SYNC signal is to align the data with local clock. The last two bits of the SYNC signal indicate the end of SYNC signal.

Packet transfer

The data is transferred in packets over the bus. This paragraph will focus on the different types of packets available.

Packet Identifier Field

This is a eight bit field with the first four bits as packet identifier and the last ones as check on correctness.

Address Field

After the PID (Packet Identifier Field) the addresses of the function field (7 bits) and endpoint.(4 bits) are presented.

Data packet

The data field may range from zero to 1,024 bytes and must be an integral number of bytes. The data is arranged by the little Endian coding.

USB Clock model

The USB has several clocks. It makes use of:

- Sample Clock: This clock determines the natural data rate of samples moving between client software on the host and the function. This clock does not need to be different between non-USB and USB implementations.
- Bus Clock: This clock runs at a 1.000 ms period (1 kHz frequency) on full-speed segments and 125.000 µs (8 kHz frequency) on high-speed segments of the bus and is indicated by the rate of SOF packets on the bus. This clock is somewhat equivalent to the 8 MHz clock in the non-USB example. In the USB case, the bus clock is often a lower-frequency clock than the sample clock,

whereas the bus clock is almost always a higher-frequency clock than the sample clock in a non-USB case.

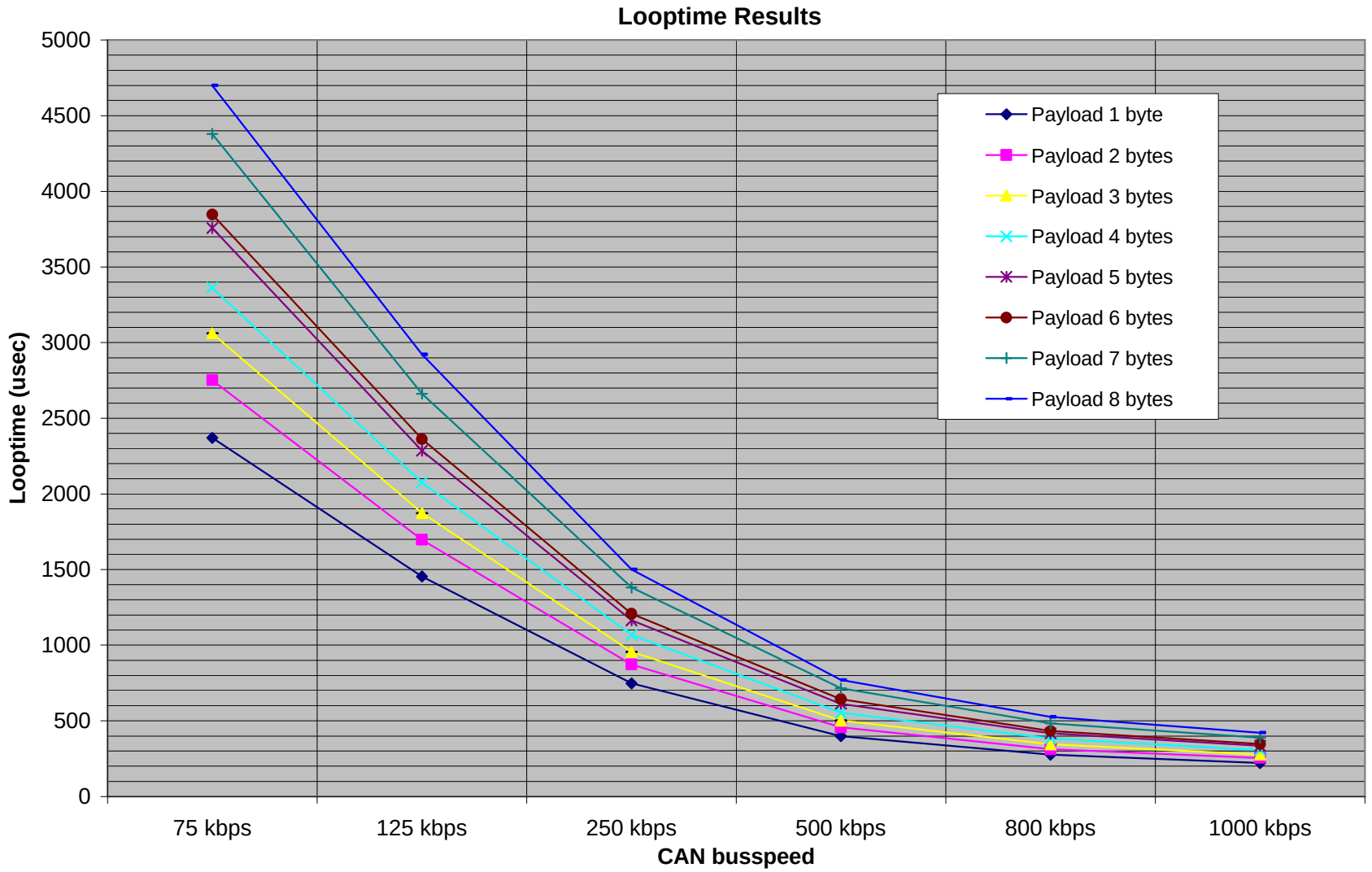
- Service Clock: This clock is determined by the rate at which client software runs to service IRP's that may have accumulated between executions. This clock also can be the same in the USB and non-USB cases.

7.2. Complete results looptime measurement

Payload	Looptime @ 75 kbps (µs)	CPU Time (µs)	Looptime @ 125 kbps (µs)	CPU Time (µs)	Looptime @ 250 kbps (µs)	CPU Time (µs)	Looptime @ 500 kbps (µs)	CPU Time (µs)	Looptime @ 800 kbps (µs)	CPU Time (µs)	Looptime @ 1000 kbps (µs)	CPU Time (µs)
Payload 1 byte	2373	11	1455	10	748	11	400	11	275	10	221	11
Payload 1 byte	2373	11	1455	10	748	11	400	11	276	11	221	10
Payload 1 byte	2368	11	1455	10	748	11	400	10	276	11	220	10
Payload 1 byte	2375	11	1455	11	748	11	400	11	275	11	221	10
Payload 1 byte	2363	10	1455	11	748	10	400	10	276	11	221	10
Payload 2 bytes	2743	11	1692	11	873	10	456	11	312	11	256	10
Payload 2 bytes	2752	11	1717	10	873	10	456	11	312	11	255	10
Payload 2 bytes	2783	10	1692	11	873	10	456	11	316	10	252	11
Payload 2 bytes	2747	11	1693	11	873	10	456	11	311	11	255	10
Payload 2 bytes	2737	11	1692	11	872	10	462	10	312	11	255	10
Payload 3 bytes	3037	11	1868	11	961	10	500	11	348	10	277	11
Payload 3 bytes	3082	10	1892	10	947	11	500	11	345	11	281	10
Payload 3 bytes	3082	10	1867	11	960	10	506	10	343	11	276	10
Payload 3 bytes	3072	10	1868	11	948	11	506	10	348	10	280	10
Payload 3 bytes	3036	11	1868	11	961	10	500	11	343	11	273	11
Payload 4 bytes	3356	11	2067	11	1080	10	557	11	381	11	305	11
Payload 4 bytes	3366	11	2105	10	1061	11	550	11	381	11	305	11
Payload 4 bytes	3366	11	2067	11	1061	11	550	11	387	10	310	10
Payload 4 bytes	3356	11	2067	11	1061	11	556	11	381	11	305	11
Payload 4 bytes	3371	11	2068	11	1078	10	556	11	387	10	305	11
Payload 5 bytes	3762	10	2255	11	1155	11	610	11	420	10	338	10
Payload 5 bytes	3748	10	2305	10	1180	10	622	10	416	11	332	11
Payload 5 bytes	3758	10	2305	10	1155	11	610	11	420	10	337	10
Payload 5 bytes	3762	10	2256	11	1155	11	608	11	412	11	335	10
Payload 5 bytes	3757	10	2306	10	1180	10	610	10	420	10	337	10

Payload	Looptime @ 75 kbps (µs)	CPU Time (µs)	Looptime @ 125 kbps (µs)	CPU Time (µs)	Looptime @ 250 kbps (µs)	CPU Time (µs)	Looptime @ 500 kbps (µs)	CPU Time (µs)	Looptime @ 800 kbps (µs)	CPU Time (µs)	Looptime @ 1000 kbps (µs)	CPU Time (µs)
Payload 6 bytes	3892	10	2342	11	1198	11	643	10	442	10	352	10
Payload 6 bytes	3818	11	2342	11	1198	11	643	10	433	10	346	11
Payload 6 bytes	3810	11	2393	10	1223	10	643	10	430	11	342	11
Payload 6 bytes	3902	10	2342	11	1222	10	643	10	430	11	346	11
Payload 6 bytes	3810	11	2392	10	1198	11	643	10	426	11	342	11
Payload 7 bytes	4481	10	2642	11	1411	10	731	10	482	11	396	11
Payload 7 bytes	4315	11	2642	11	1361	11	706	11	482	11	382	11
Payload 7 bytes	4475	10	2742	10	1361	11	706	11	482	11	396	10
Payload 7 bytes	4315	11	2642	11	1411	10	731	10	482	11	383	11
Payload 7 bytes	4311	11	2642	11	1361	11	706	11	482	11	386	11
Payload 8 bytes	4671	11	2856	11	1467	11	793	10	521	11	416	11
Payload 8 bytes	4655	11	2855	11	1466	11	766	11	521	11	430	10
Payload 8 bytes	4665	11	2967	10	1522	10	765	11	535	10	416	11
Payload 8 bytes	4660	11	2967	10	1523	10	760	11	517	11	416	11
Payload 8 bytes	4846	10	2967	10	1523	10	766	11	536	10	430	10

7.3 Graph looptime measurements



7.4 Manual: Getting started

Introduction

This Appendix is a short manual on how to get started on using the Tasking compiler in combination with the two IME ACTIA Development boards and the PCSLIMLINE CAN isa card.

Hardware checklist

Before starting to setup the boards and software, the following items are needed:

- PC with a Windows operating system and one empty isa slot.
- PCSLIMLINE Isa Card
- 2 IME ACTIA Development Boards
- 2 RS232 cables
- 1 CAN cable
- 2 voltage sources 12 V DC
- Software: Tasking C166/ST10 Version 7.0 install files
- Software: ASM files for the IME ACTIA boards: cstart.asm, reg167.h, 167creg.h
- CDROM with IME ACTIA CAN Products and Tools
- Manuals of all boards and datasheets of the Infineon C167CR
- If used: project files saved during the individual design project: 'real time control on CAN'

If all components are available, the development can start.

PCSLIMLINE ISA CARD

The pcslimline isa card should be installed in a empty slot. The setings for Irq and memory address should be remembered for software istallation.

Use the CDROM provided by IME ACTIA to install the drivers. After launching the CD, choose *pcslimline (levelx) driver V1.06** to install. After selecting and launching a password will be asked. The password is : **wvu0cnyy**.

After installation the slimline card can be found in the configurationmanager at hardware. If the card isn't functioning properly, check the irq and memory address there.

IME ACTIA Development boards

The development boards should be connected to their powersupplies and both serial cables should be connected to pc and boards. The CAN bus cable can be connected to PC and the two boards. After setting up, the software must be installed. The software contains of three parts: levelx drivers, pccancontrol and hexload. All three parts can be found on the CDROM from IME ACTIA. Each program has its own password:

- C167CR EvaBoard (Levelx) serial driver V1.04: **1yc+0h51**
- PC CANControl V1.06: **ij/iint**
- C167CR Evaluation Board Firmware samples + Loader: **gtwjxv9u**

After installing both isa card and development boards their functionality can be tested with PCCANControl. When running the program, a project must be selected. In this project hardware must be

selected. The PCSlimline should work always, when selected. The Evaboard #1 or #2 (the number corresponds to the serial port to which the board is connected) only works when powered up and when the board is in *bootstrap mode*.

Bootstrap mode

The Evaboards can be run in bootstrap mode or normal mode. In normal mode, the boards starts up from the onboard eeproms and runs the program which is in the eeproms. In bootstrap mode the board can be programmed through RS232. In this mode the pc can control the board. PCCANControl, the tasking hardware debugger and hexload all use the board in bootstrap mode. Bootstrap mode can be activated by setting JP21 on the EVABoard (to be found in the row next to the LED array , jumper closest to the LED array). See also board layout in the manual.

Hexload

When running the boards in bootstrap mode it is possible to download hex files into the RAM of the board with the help of the program Hexload. This program runs in a dosbox. The syntax of the program is:

HEXLOAD [command] <filename.hex>

[command]: COM=#, defines the serial port whereto the board is connected. (default=1)
RATE=#, defines the speed of the serial port. (default=19200)

After downloading the hex-file the boards starts executing the program. If power is disconnected or the reset button is pushed(see manual) the memory is cleared and the program has to be downloaded again.

TASKING C166 Development Tools
After installing the TASKING C166 compiler, it is possible to make your own program. The steps to make get to the the final compilation of a hex file, the following steps have to be done:

1. Create a project space: a project space can contain more projects.
2. Create a project: is a member of a projectspace
3. Add standard files to a project
4. Set-up the compiler
5. Set-up the Crossview debugger

Each step will be discussed now.

1. Create a Workspace:

After starting the C166 compiler, first a project space has to be opened. Figure 1 displays the menu where to create one: Project -> Project Space -> New
By clicking this option a name for the project spaced will be asked.

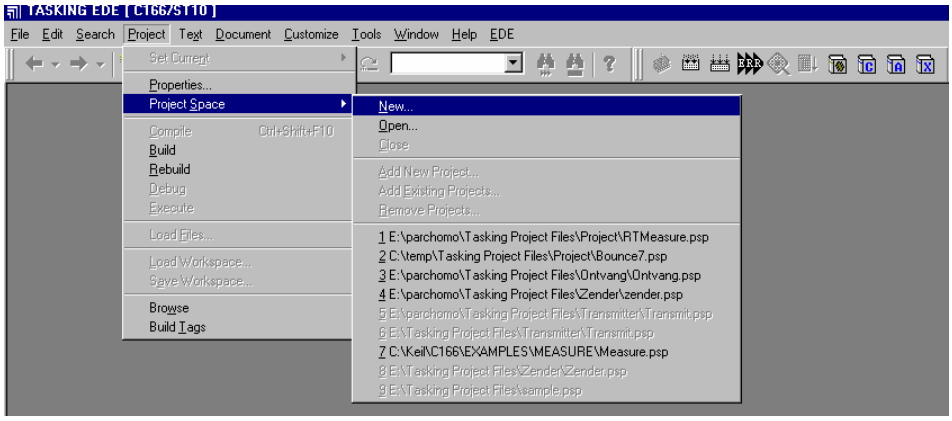


Fig 1: creating a projectspace

After filling out a name, step 2 has to be followed.

2. Creating a project

Figure 2 shows the menu for creating a project in a workspace. By clicking the  icon, a new project can be added.

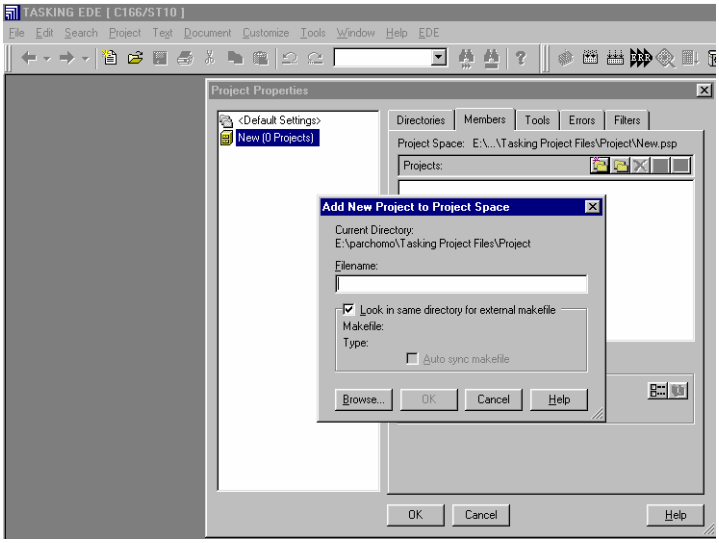


Fig 2: creating a project

To every project standard files have to be added. The standard files consist of registry definitions and startup code needed for correct compiling. Adding these files is described in 3.

3. Adding standard files.

The files cstart.asm, reg167.h and C167reg.h have to be added to each project. Adding the files is done by clicking the  icon. Figure 3: shows the adding of files.

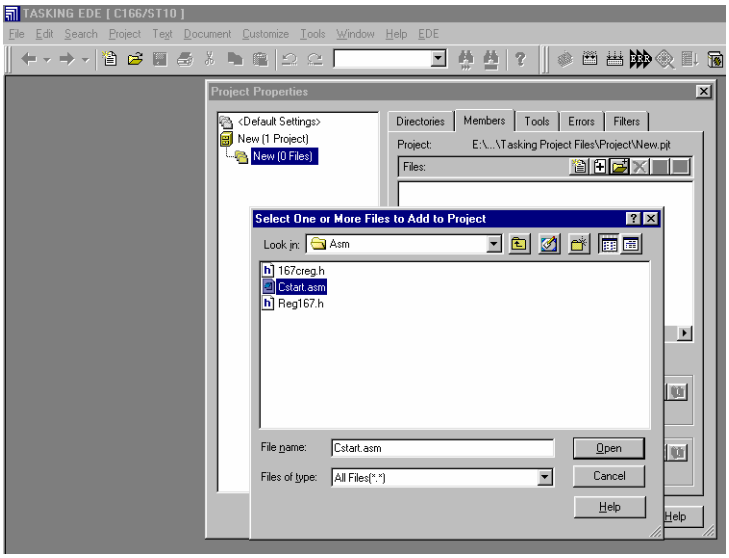


Fig 3: Adding files to project

After adding all three files Tasking creates an environment as can be found in figure 4. For programming in C a new C-file has to be made. This can be done through File -> New -> 'filename.c'. This file is automatically added to the project. Now only the debugger and the compiler output has to be set. This is described in steps 4 and 5.

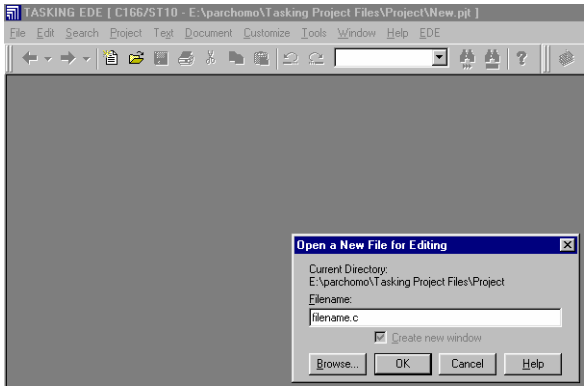


Figure 4: creating a c file.

4. Setting up compiler

The only thing to be done to set the compiler right is to set the output to hex-file. Without this no hex-file will be created and thus it won't be possible to download a self-made creation in the development board. Creating the hex-file is set through EDE -> Linker/Locater options ->check Intel HEX records for eprom programmers. See also figure 5.

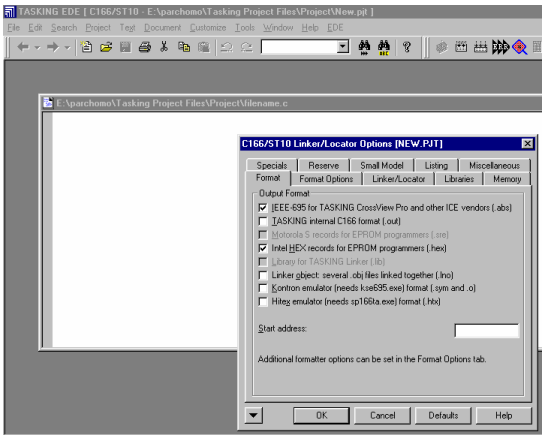


Figure 5: setting hex-file output

Now you can compile your project by clicking the  icon.

5. Setting the debugger

There are to ways to test the made application: through the simulator or by means of hardware debugging. The simulator is set default so, when debugging the application (by clicking the  icon), the simulator is started automatically.

To use the hardware debugger the options in crossview have to be set as follows:
EDE -> CrossviewPro options -> Rom/RAM Monitor
At Target choose: *I+ME C167C.* In figure 6 this option is also showed. When set the debugger will load a monitor program in the board connected to COM 1 (this can be changed at EDE -> CrossviewPro options ->communications), followed by the compiled application. By running the application in Crossview Pro the application can be tested. Breakpoints can be inserted and the application can also be runned per line of c-code.
Please be sure to set the Evaboard to *Bootstrap mode*!

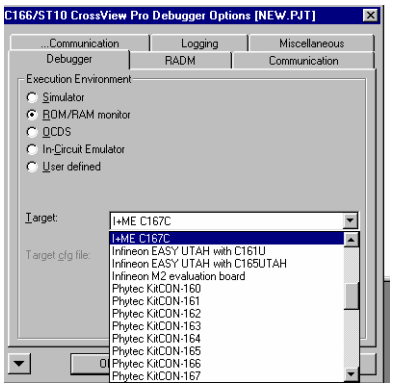


Figure 6: Setting the hardware debugger

This concludes the first steps to get started with Tasking and CAN boards. Please note that instead of creating a new project it is also possible to load an existing project. In this case it is only to check steps 4 and 5 for correct functionality of compiler and debugger.

7.5 References

Actia, I. (2002), *PC CANCONTROL*,http://www.ime-actia.com/pccancontrol.htm,

Automation, C.i. (2002), *Can in Automation Homepage*,http://www.can-cia.de,

Engineering, C. (2002), *Control Engineering website*,http://www.ce.utwente.nl/RTweb/research/annual_report_2000/Embedded_Control_Systems.htm,

Keil (2002), *Keil Software Development Tools for the 8051,251 ARM7 and C16x/ST10 Microcontroller*,http://www.keil.com,

Tasking (2002), *Embedded Software Development from Altium*,http://.www.tasking.com,

Schofield, M. (2001), *Controller Area Network, An introduction to CAN - The serial data communications bus*,http://www-ife.tu-graz.ac.at/Lokal/Tech/Can-Bus/schofield.htm,

Führer, T. and B. Müller (2000), *Time Triggered Can*,http://www.can.bosch.com/content/Literature.html,

USB.org (2000), *USB 2.0 specifications*,http://www.usb.org/developers/data/usb_20.zip,

CIA, C.i.A. (1999a), *CAN Application Layer*,http://www.can-cia.de/CANappl.pdf,

CIA, C.i.A. (1999b), *CAN Datalink layer*,http://www.can-cia.de/CANdll.pdf,

CIA, C.i.A. (1999c), *CAN Physical Layer*,http://www.can-cia.de/CANphy.pdf,

Kopetz, H. (1997), *Real-Time Systems, Design principles for Distributed Embedded Applications*, Kluwer Academic Publishers, Boston, 0-7923-9894-7.